VEHICLE ROUTING AND SCHEDULING IN A MULTI - PRODUCT DISTRIBUTED MANUFACTURING SYSTEM

by

V. SUNIL

DEPARTMENT OF INDUSTRIAL & MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MARCH, 1997

VEHICLE ROUTING AND SCHEDULING IN A MULTI-PRODUCT DISTRIBUTED MANUFACTURING SYSTEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

V.SUNIL

to the

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

MARCH,1997

CERTIFICATE

It is to certify that the work contained in the thesis entitled " VEHICLE ROUTING AND SCHEDULING IN A MULTI-PRODUCT DISTRIBUTED MANUFACTURING SYSTEM " by Mr. V.Sunil has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

28 Feb., 1997.

Prof. A. K. Mittal
Industrial Management and Engineering Department
Indian Institute of Technology
Kanpur - 208016.

ACKNOWLEDGMENT

I would like to express my sincere thanks to my thesis supervisor Prof.A.K.Mittal. Despite his demanding schedule, he made himself available whenever I required his guidance. The many discussions that I had with him channelised my thinking and helped me in drawing meaningful conclusions. I am indebted to him for making this research endeavor fruitful.

I am thankful to IME colleagues, for their invaluable support in carrying out this work. I am grateful to my teachers, Dr.TP Bagchi, Dr. RK Ahuja, Dr Kripa Shankar, Dr AP Sinha, Dr. Rahul Varman and Dr. RRK. Sharma for their constant support.

I am grateful to my parents whose constant encouragement made me to fulfill my goals.

I sincerely thank the other IME family members for their help and encouragement, provided to me during my Master's program.

March, 1997 V.Sunil

ABSTRACT

In this dissertation an attempt has been made to solve a problem of vehicle routing and scheduling in a multi-product distributed manufacturing environment. It is being assumed that processing facilities are distributed geographically and the material has to be transported between these processing centres according to their product processing sequence or alternatively transportation between process centres are significant in comparison for process time.

To solve this problem, we formulate the problem as a mixed integer program. As the problem size increases it is computationally difficult to solve the problem with exact algorithms and the problem is NP-hard. Given the intrinsic difficulty of these class of problems, heuristic methods seem to offer the most promise for practical size problems We propose two types of heuristic algorithms to solve this problem based on sequential and parallel insertion of the tasks. We also test a few despatch rules in assigning the tasks to the vehicles. The performance of heuristics is compared computationally. The test problem set includes routing and scheduling environments that differ in terms of the type of data to generate the problems, the percentage utilisation of process centres, slack time available for the products, processing time to travelling time ratio and the problem size parameters. Heuristics are also modified and performance studied to situations where the number of vehicles is restricted, and where the violation of due-dates of the products are allowed by the inclusion of delay costs.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Vehicle routing and scheduling :

Transportation comprises a significant fraction of the economy of most developed nations. For example, a National Council of physical Distribution Study [1978] estimates that transportation accounts for 15% of the U.S. gross national product. This economic importance has motivated both private companies and academic researchers to vigorously pursue the use of operations research and management science to improve the efficiency of transportation. [3]

Various modes of transportation exist including air, rail, ship and motor vehicle. The research on transportation has focused on different issues in each mode. In air, the efficient schedule of airline crews [Lavoi, Minous & Odier, 1988] has received primary attention while in rail, use of large-scale real time computer control systems has dominated the research agenda (e.g. Harker 1990). The literature on both motor vehicles (trucks, school buses and general passenger buses) and ships has focused on a common problem -- the efficient use of a fleet of vehicles that must make number of stops to pick up and/or deliver passengers or products. The problem requires one to specify which customers/products should be delivered by each vehicle and in what order so as to minimise total cost subject to a variety of constraints such as vehicle capacity and delivery time restrictions. [3]

The work in this dissertation is motivated by a vehicle routing and scheduling problem in a geographically distributed manufacturing system, for an multi - product or jobbing type of organisation. This problem resembles the Pickup - Delivery problem with time windows (PDPTW) and Advance Request Dial-A-Ride with Time Windows (ADARTW) with additional product sequence constraints (more details in section 2.1). In this dissertation we will concentrate on, minimisation of the total transportation cost involved in transporting the products between the processing centres according to their product sequence, without violating the time windows.

1.2 Literature review :

Laporte etal. [1994], stated in his survey that "Over the past three decades, operation researchers interested in vehicle routing and scheduling have emphasised in the development of algorithms for pure vehicle routing problems (Mingozzi, and Toth 1979, Bodin, Golden, Assad, and Ball 1983, Laporte 1992) and vehicle routing problems with time windows (Desrosiers, Soumis 1985 , Solomon 1987)". This section provide a brief review of the literature available to solve different vehicle routing problems with time windows.

In this literature survey, we considered two types of problems. One is the *vehicle routing and scheduling problem with time windows* (VRSPTW), which defined as follows. A number of vehicles are located at a single depot and must serve a number of geographically dispersed customers. Each vehicle has a given capacity. Each customer has a given demand and must be served within a specified time window. The objective is to minimise the total cost of travel [1].

The second problem type is the *pickup and delivery problem with time windows* (PDPTW). Again, there is a single depot, a number of vehicles with given capacities, and a number of customers with given demands. Each customer must now be picked up at his origin during a specified time window, and delivered to his destination during another specified time window. The objective is to minimise total travel cost. [1]

The special case in which all customers demands are equal is called the *dial-a-ride problem* (DARP). It arises in transportation systems for the handicapped and the elderly In these situations, the temporal constraints imposed by the customers strongly restrict the total vehicle load at any point of time, and the capacity constraints are of secondary importance. The cost of route combination is a combination of travel time and customer dissatisfaction. [1]

Christofides, Mingozzi and Toth [1981] proposed a dynamic programming algorithm to solve the single vehicle problems with time windows. Such a dynamic programming approach can be useful for several NP-hard routing problems. However, the cardinality of the state space is usually exponential in the problem size. The practical use of dynamic programming in this context is restricted to state spaces of at most pseudo polynomial size and relatively small problem instances.

Desrosiers, Dumas and Soumis [1986] give a similar 2n-stage dynamic programming algorithm for a capacitated single-vehicle PDPTW. They proposed number of state elimination rules to reduce the computational effort. This algorithm can solve real-life problems with up to 40 customers.

Dumas, Desrosiers and Soumis [1991] proposed an exact algorithm for a multiple vehicle pickup-delivery problem with time windows. This algorithm is based on a Dantzig-Wolfe decomposition approach embedded into a branch-and-bound search tree. The master problem results in the linear relaxation of a set partitioning type model, while feasible routes or columns are generated by a sub problem modelled as a shortest path problem with coupling, precedence, time window and capacity constraints. The algorithm has been successful in solving two real life problems of sizes 19 and 30 requests, respectively. Very good solutions were obtained within the exploration of a few nodes in the branch-and-bound tree, typically less than 10. The algorithm is appropriate for the PDPTW when the problem solutions require an average up to 5 requests per vehicle, i.e., 10 pickup and delivery stops. However it becomes ineffective in the dial-a-ride context due to the actual size of practical problems have a large number of requests per vehicle.

Marius M. Solomon [1987] had applied Clarke and Wright's [1964] Savings heuristic, Time oriented Nearest-Neighbour heuristic, Time oriented sweep heuristic (original sweep heuristic was proposed by Gillett and Miller [1974]) to VRSPTW. He also proposed an Insertion type of heuristic for this problem. Through his extensive computational study for different type of problems, he concluded that the insertion type of heuristics proved to be very successful.

J.J.Jaw, Odoni, Psaraftis and Wilson [1986] had proposed an algorithm for an Advance Request Dial-A-Ride Problem with time windows (ADARTW). In this problem, "advance - request" means that all the requests are received well before the time of vehicle dispatching. The algorithm uses a sequential insertion procedure to assign customers to vehicles and to determine a time schedule of pick-ups and deliveries for each vehicle.

1.3 Present work :

In the literature surveyed no work had been done for a PDPTW with additional sequence constraint. Hence as stated earlier the work in this dissertation mainly deals with a PDPTW with a sequence constraint case in a multi-product distributed manufacturing system.

In chapter 2 we define the problem formulate this problem as a conventional Pickup-Delivery Problem with Time Windows (PDPTW) with additional product sequence constraint. In this chapter we also describe the Integer programming formulation for this problem. In chapter 3 two major heuristic approaches proposed for this problem are described. Further, few despatch rules for the selection of the task to be assigned to the vehicle are discussed. We also describe modified heuristics for the case, when the number of vehicles is restricted. In chapter 4 we discuss the design of the problems and study the performance of the heuristics and despatch rules for various problems designed. Chapter 5 concludes taking a retrospective view on the work done in this dissertation and suggests logical extension to the problem analysed for further research.

CHAPTER 2

PROBLEM DEFINITION , ANALYSIS AND FORMULATION

2.1 Problem definition :

Unlike conventional manufacturing systems, in this thesis it is being assumed that processing facilities are distributed geographically and the material has to be transported between these processing centres. Such an system implies that, travel times are atleast comparable to the processing times and hence transportation schedule and times will have impact on the completion time of the job. Further travel costs are significant and will have impact over production cost. While such a situation will become relevant in truly global manufacturing scenario. it is also meaningful situation in ease of conventional plants when batch sizes are small and hence the processing times are small compare to travel time. This also may be relevant in a JIT implementation for a multi-product situation, where large number of products are required to be produced although in small batches.

In this thesis, we consider a hypothetical distributed manufacturing system, for an multi-product or jobbing type of organisation. In this problem, each batch of product has a pre-specified definite processing sequence and due-dates. The processing times at different processing centres for each batch of product are also pre-specified. After processing a batch of product at a processing centre according to the processing sequence of that product they should be transferred to the next processing centre for further processing if any.

The objective of the transportation system is to perform all these pickup-delivery requirements in time (i.e., not exceeding the specified due-date), with minimum total travelling cost . We will also consider the case, where the number of vehicles are limited. In such a situation, attempt will be to minimise travelling and delay cost.

Here , the following *assumptions* are taken into account :

(1) Vehicle depot is at equal distance from all the processing centres .

(2) All the vehicles have equal capacity .

(3) All the vehicles have uniform travelling speed .

(4) Loading and Unloading times are negligible (or) are absorbed in travel times.

(5) More than one batch of product can be simultaneously processed at a processing centre (or) scheduling delays are absorbed in process times.

(6) Production capacity are sufficiently large and hence there are no schedule delays at the process centres (or) they are absorbed in the process times.

2.2 Problem analysis :

Let there be 'n_p' number of batches of products and 'n_d' number of processing centres in a manufacturing system. A product batch 'i' is to be processed in 'm_i' number of stages. At each stage it is to be processed in a specified processing centre. From the processing time, travelling time and product batch sequence data, the time windows of all pickup and delivery tasks at all stages can be found for each batch of product as follows :

Notations :

EPT (i , j) $\rightarrow$ Earliest pickup time for i_{th} batch of product at j_{th} stage .

EDT (i , j) $\rightarrow$ Earliest delivery time for i_{th} batch of product at j_{th} stage .

LPT (i , j) $\rightarrow$ Latest pickup time for i_{th} batch of product at j_{th} stage .

LDT (i , j) $\rightarrow$ Latest delivery time for i^{th} batch of product at j_{th} stage .

Dd_i $\rightarrow$ Due-date for i^{th} batch of product.

Sl_i $\rightarrow$ Total slack time available for i_{th} batch of product .

μ_{ik} $\rightarrow$ Processing centre for i^{th} batch of product at k^{th} stage.

t1 (i , k) $\rightarrow$ Processing time for i_{th} batch of product at k_{th} stage .

t2 (μ_{ik} , μ_{ik+1}) $\rightarrow$ Travelling time from the processing centre at k_{th} stage to processing centre at $(k+1)_{th}$ stage for i_{th} batch of product .

n_i^+ $\rightarrow$ Total number of pickup nodes for i^{th} batch of product.

$$n_i^+ = (m_i - 1)$$

n_i^- $\rightarrow$ Total number of delivery nodes for i^{th} batch of product.

$$n_i^- = (m_i - 1)$$

Total number of pickup & delivery nodes for batch i $= 2n_i$

Total number of pickup nodes for all n_p batches n $= \sum_{i=1}^{n_p} n_i$

Total number of delivery nodes for all n_p batches n $= \sum_{i=1}^{n_p} n_i$

Total number of pickup & delivery nodes for all n_p batches $= 2n$

2.2.1 Calculation of time-windows for pickup-delivery nodes:

$$Sl_i = Dd_i - \left(\sum_{k=1}^{n_i} t1(i,k) + \sum_{k=1}^{n_i} t2(\mu_{ik}, \mu_{ik+1}) \right)$$

$$EPT(i,j) = \underbrace{\sum_{k=1}^{j-1} t1(i,k)}_{(1)} + \underbrace{\sum_{k=1}^{j-1} t2(\mu_{ik}, \mu_{ik+1})}_{(2)}$$

| (1)- Sum of proc.
| times of previous
| stages .
|
| (2)- Sum of min.
| travelling times .

$$LPT(i,j) = EPT(i,j) + Sl_i$$

$$EDT(i,j) = EPT(i,j) + t1(i,j)$$

$$LDT(i,j) = EDT(i,j) + Sl_i$$

After the calculation of time windows, the present problem resembles the standard pickup - delivery problem with time windows, with additional constraint that pickup and deliveries have to be made in specified order for each product.

In this problem, the objective is to minimise the total cost involved in performing all the pickup - delivery requirements within the specified time windows .

The present problem can be better illustrated through the following example.

2.2.2 Example :

Let there are two batches of products and each product has three operations. Processing sequence, processing times, due dates, vehicle capacity, vehicle speed and distance between the processing centres are as follows.

	Product 1			Product 2		
Sequence	MC1 → MC3 → MC2			MC2 → MC1 → MC3		
Proc time	3	2	1	2	3	2
Due-date	15			18		

Vehicle capacity = 3 unit loads   Vehicle speed = 50 kmph

Distance between processing centres :

	MC1	MC2	MC3
MC1	0	75	125
MC2	75	0	100
MC3	125	100	0

Now, by using the above data we can calculate the time windows for each pickup-delivery task as follows :

Slack available for 1st batch of product $Sl_1 = 15-((3+2+1)+(2.5+2))=15-10.5=4.5$

Similarly, for the second product $Sl_2 = 7$

Time window for the first p-d task of the first product:

EPT(1, 1) = 3   LPT(1, 1) = 3+4.5 = 7.5

LPT(1, 1) = 3+2.5 = 5.5   LDT(1, 1) = 5.5+4.5 = 10

Similarly, for all other pickup-delivery tasks time windows are calculated. Now, the transportation requests are as follows:

Pickup Product1 MC1 (3, 7.5)	Delivery Product1 MC3 (5.5, 10)	Pickup Product1 MC3 (7.5, 12)	Delivery Product1 MC2 (9.5, 14)
Pickup Product2 MC2 (2, 9)	Delivery Product2 MC1 (3.5, 10.5)	Pickup Product2 MC1 (6.5, 13.5)	Delivery Product2 MC3 (9, 16)

Now the objective of the transportation system is to meet all these requests at minimum total travelling cost, without violating vehicle capacity, time feasibility and product sequence constraints. The present problem can be modelled as a network flow problem. In this network pij & dij represents pickup and delivery nodes of ith product at jth stage. Here, an arc exists between two nodes, only if it satisfies time feasibility and product sequence constraints and also all the pickup nodes will have an incoming arc from depot and all delivery nodes will have an out going arc to the depot. For the example problem the network will be as shown in figure 2.1.

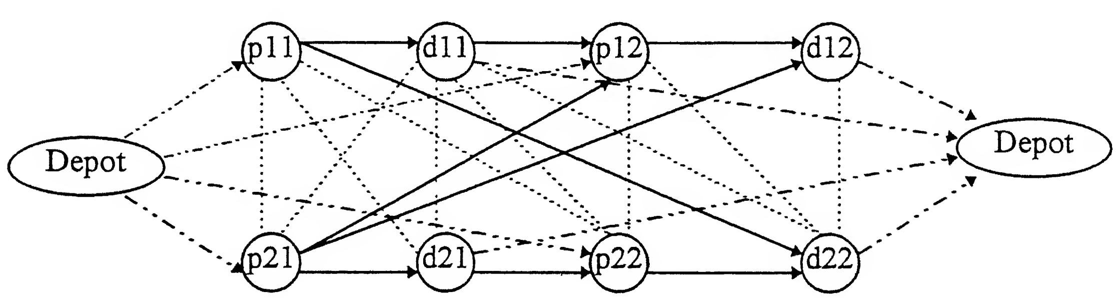


Fig 2.1 : Network Flow Model

Directed arc from (or) to the depot	- · · - - - · →
Undirected arc	···············
Directed arcs between nodes	———→

Now, the objective is to find out the optimal routing of the vehicles from the depot, which minimises the total transportation cost involved. The constraints in this problem are:

- a node should be visited by only one vehicle that too only once.
- a node should be visited within the specified time window.
- a vehicle can visit a delivery node only after visiting the corresponding pickup node.
- the vehicle which visits a pickup node (say p21) should visit the corresponding deliverynode also (i.e., d21).
- a vehicle can visit a higher stage node (say p12) of a product, only after the completion of lower stage tasks. of that product (i.e., p11-d11).
- the vehicle load should not be more than its capacity.

2.3 Mathematical formulation of the problem :

Let there be *n* pickup - delivery tasks to be performed including all batches of products . Then the total number of pickups and deliveries is equal to 2*n* . Then the different sets of nodes in the network are represented as follows :

Set of all nodes in the network $N = \{ 0 , 1 , 2 , \ldots , n-1 , n , n+1 , \ldots , 2n , 2n+1 \}$

Here , node '0' and '2n+1' represents the depot .

Set of all pickup nodes in the network $N^P = \{ 1 , 2 , 3 , \ldots , n \}$

Set of all delivery nodes in the network $N^D = \{ n+1 , n+2 , \ldots , 2n \}$

Let there be n_p batches of products , then

Set of all pickup nodes for 1st batch of product $N_1^P = \{1, 2, \ldots, n_1\}$

Set of all delivery nodes for 1st batch of product $N_1^D = \{n+1, n+2, \ldots, n+n_1\}$

Similarly for all other (k-1) batches of products we can represent the pickup and delivery nodes through systematic node numbering .

et_i $\rightarrow$ Earliest pickup (or) delivery time at node 'i' .

lt_i $\rightarrow$ Latest pickup (or) delivery time at node 'i' .

q_i $\rightarrow$ Load to be pickedup or delivered at a node

Notations and variables used in the mathemeatical formulation :

x^v_{ij} $\rightarrow$ Binary variable $\forall\, v \in \{1,2,\ldots,n_p\}$, $i \in \{1,2,\ldots,n\}$ & $j \in \{1,2,\ldots,n\}$

$x^v_{ij} = 1$ when vehicle 'v' travels from node 'i' to node 'j' .

$x^v_{ij} = 0$ Other wise .

at_i $\rightarrow$ Actual pickup (or) delivery time at node 'i' .

Ld^v_i $\rightarrow$ Load of vehicle 'v' at node 'i' .

Q $\rightarrow$ Capacity of a vehicle .

C_{ij} $\rightarrow$ Cost involved in travelling from processing centre i to processing centre j.

F $\rightarrow$ Fixed cost per vehicle.

Formulation :

objective function - Min $\sum_{v=1}^{n} \sum_{i=1}^{2n} \sum_{j=1}^{2n} x^v_{ij} \cdot C_{ij} + \sum_{v=1}^{n} \sum_{j=1}^{n} x^v_{0j} * F$

subjected to ---

$$\sum_{v=1}^{n} \sum_{j=1}^{2n} x^v_{ij} = 1 , \ \forall \ i \in \{ N^P \cup N^D \} \longrightarrow (1)$$

$$\sum_{j=1}^{2n} x^v_{ij} - \sum_{j=1}^{2n} x^v_{ji} = 0 , \forall i \in \{ N^P \cup N^D \} , v \in [1,n] \longrightarrow (2)$$

$$\sum_{j=1}^{n} x^v_{0j} \leq 1 , v \in [1,n] \longrightarrow (3)$$

$$\sum_{j=1}^{n} x^v_{j(2n+1)} \leq 1 , v \in [1,n] \longrightarrow (4)$$

$$\sum_{j=1}^{2n} x^v_{ij} - \sum_{j=1}^{2n} x^v_{j(n+i)} = 0 , \forall i \in N^P , v \in [1,n] \longrightarrow (5)$$

$$at_i + t1 (\mu_i , \mu_{n+i}) \leq at_{(n+i)} \ \forall i \in N^P \longrightarrow (6)$$

$$at_{(n+i)} + t1 (\mu_{n+i} , \mu_{i+1}) \leq at_{(i+1)} \forall i \in N_k^P , k \in [1,n_p] \longrightarrow (7)$$

$$at_j + (1-x^v_{ij}) * M \geq at_i + t2(\mu_i , \mu_j) \forall i , j \in \{ N^P \cup N^D \} \ \& \ v \in [1,n] \longrightarrow (8)$$

$$et_i \leq at_i \leq lt_i \ \forall i \in \{ N^P \cup N^D \} \longrightarrow (9)$$

$$Ld^v_j = Ld^v_i + q_j * x^v_{ij} \forall i , j \in \{ N^P \cup N^D \} \ \& \ v \in [1,n] \longrightarrow (10)$$

$$Ld^v_0 = 0 \ \forall v \in [1,n] \longrightarrow (11)$$

$$Ld^v_i \leq Q \ \forall i \in N^P \longrightarrow (12)$$

$$x^v_{ij} \in \{ 0 , 1 \} \forall i , j \in N , v \in [1,n] \longrightarrow (13)$$

Here, C_{ij} represents the cost involved in travelling from node i to node j and F is the fixed cost incurred by using a new vehicle from the depot and M is a very large number. The objective function is minimisation of the sum of these costs .

Constraint 1 ensure that only one vehicle visits a node and that too only once. *Constraint 2* ensure that if a vehicle visits a particular node it should leave that node. *Constraint 3* ensure that a vehicle should leave the depot only once and *constraint 4* ensure that a vehicle should arrive the depot only once. *Constraint 5* ensure that a vehicle which visites the pickup node, that should visit the corresponding delivery node also. *Constraint 6* ensure that the vehicle should load first at the pickup node and then unload at the corresponding delivery node *Constraint 7* ensure the product sequence constraint (i e., only after completion of the previous pickup-delivery task, the next p-d task for that batch of product can be started) *Constraint 8* calculates the actual pickup and delivery times after visiting the node . *Constraint 9* ensure the actual time of visiting is within the time-windows . *Constraints 10 , 11 & 12* ensure the capacity constraint of the vehicles . *Consraint 13* ensure non-negativity of variables

In the above integer programming formulation, the estimated number of variables are of $O(n^3)$, where 'n' is the total number of pickup & delivery nodes. The exact details about the number of variables are as follows :

- Number of binary variables (i.e., X^v_{ij} 's) $= (n^3 + n^2) / 2$
- Number of vehicle load variables (i.e., Ld^v_1 's) $= n^2/2$
- Number of time variables (i.e., at_i 's) $= n$
- Total number of variables $= (n^3/2 + n^2 + n)$

The total number of constraints in this formulation are ($8n^2 + 9n$).

Marius M. Solomon [1987] stated that problems with time windows are frome the computational complexity perspective quite difficult. Since the VRP is NP hard, by restriction, the VRSPTW is NP hard. Further more, Savelsberge [1984] has shown that even finding a feasible solution to the VRSPTW when the number of vehicles is fixed is itself a NP-complete problem. Therefore, the development of heuristic algorithms for this problem class are primary interest.

Yvan Dumas et al. [1991] had proposed an exact algorithm to solve this type of pickup - delivery problem with only time window constraint (PDPTW). In their algorithm, they used a column generation scheme, with a constrained shortest path as a sub problem. Their algorithm worked well for tight capacity and loose time window constraints. The maximum size of the problem they solved was having about 50 transportation requests.

In spite of recent success of optimisation algorithms for vehicle routing with time windows, it is unlikely that they will be able to solve large scale practical problems. In many situations one has to settle for algorithms that run fast but may produce suboptimal solutions. So, in this dissertation we made an attempt to develop heuristics for this problem, which can give suboptimal solutions for large scale problems.

CHAPTER 3

SEQUENTIAL AND PARALLEL INSERTION HEURISTIC ALGORITHMS

In this chapter we will discuss two heuristic methods to solve the problem outlined in the previous chapter. The two algorithms differ from each other primarily in the fashion in which tasks are assigned to the vehicles. In the first approach, the tasks are continued to be assigned to a vehicle till no task can be further assigned to it. Only after that the next vehicle is picked up for loading. Sequential insertion algorithm uses this approach. In the second approach, the tasks are assigned to the best suited vehicle among the vehicles available. Parallel insertion algorithm uses this approach. In the section 3.1 we will describe in detail the working of Sequential Insertion algorithm. In this section we will also describe all the common features for both the algorithms. In the section 3.2 we will describe the Parallel Insertion algorithm. In the section 3.3 we will describe the heuristics with simple despatch rules applied for both the Sequential and Parallel insertion case for the task selection. In the last section 3.4 we will describe the algorithms to consider the case where number of vehicles is restricted.

Before describing the algorithms, we will define some of the terms used.

<u>Definitions:</u>

1) Pickup-delivery task (p-d task) :

   The activity of loading a batch of product at a process centre and transporting those products to another process centre and unloading that batch is called a pickup-delivery task.

2) Sequential Feasible tasks :

Those tasks for which all the preceding tasks have been completed are called sequence feasible tasks.

3) Feasible insertion combination of a task :

The insertion combination of the pickup and delivery nodes of a task is feasible only if

(a) it is sequence feasible

(b) it does not exceed vehicle capacity

(c) it does not exceed the latest delivery time of that task and also the latest delivery times of the tasks which are already in the travel schedule of a vehicle

4) Feasible task :

A task is said to be feasible task to assign, to a travel schedule of a vehicle only if it has at least one feasible task insertion combination.

3.1 SEQUENTIAL INSERTION ALGORITHM (SEQIA):

3.1.1 Overview of the algorithm :

In this algorithm pickup-delivery tasks are assigned to the travel schedule of the first vehicle till there is no feasible task, which can be assigned to that vehicle. After this the algorithm starts assigning tasks to the next vehicle. Thus this algorithm proceeds, by sequentially assigning tasks to the vehicles, until all the transportation requests are processed. Central to the algorithm are:

(i) *a search for feasible insertions of tasks into travel schedules*

(ii) *an optimisation step to find the best feasible insertion*

- Insertion step:

  An insertion of a task into the travel-schedule is feasible only if it does not lead to the violation of vehicle capacity constraint , time feasibility constraint and process sequence constraint for the newly assigned task and for all other tasks already assigned to that vehicle.

- Optimisation step:

  The optimisation step deals with minimising the incremental cost due to insertion of the new task into a vehicle travel-schedule. The cost function used is the sum of tangible cost due to increase in travel distance and intangible costs due to the increase in slack utilisation and waiting time(more details in section 3.1.4)

Step by step procedure of this algorithm is illustrated through flowchart in figure 3.1.1

Notations :

FTS -- Feasible task set.

It stores all the process sequence feasible tasks details at every stage of the algorithm. For each task it stores the earliest and latest pickup-delivery times and also the pickup and delivery process centre numbers.

VAS(k) -- Vehicle activity set for vehicle 'k'.

It stores the details of the travel schedule of vehicle 'k' at all stages of the algorithm. For each task the following details are stored:

- Actual time of occurrence of that task.
- Process centre at which that task occurs.
- Maximum possible shift[1] for that task.

n	-- Total number of the products.
P_{ik}	-- Pickup process centre number of k^{th} stage task of i^{th} product.
D_{ik}	-- Delivery process centre number of k^{th} stage task of i^{th} product.
Prt (i, k)	-- Processing time of k^{th} stage task of i^{th} product.
Trt(d_1 , d_2)	-- Travelling time from process centre d_1 to process centre d_2.
Dist(d_1 , d_2)	-- Distance from process centre d_1 to process centre d_2.
Ctr	-- Travelling cost per unit distance travel.
Csl	-- Slack utilisation cost per unit slack time utilised for a product.
Cwt	-- Waiting cost per unit waiting time utilisation.
Cpen	-- Penalty cost per unit time violation of due-date for a product
FC	-- Fixed cost per vehicle.
Slu(i)	-- Product i's total slack time utilised due to an insertion of new task into the travel schedule.
PSlu(i) & DSlu(i)	-- Product i's slack time utilised due to new pickup node and delivery node insertion respectively
Sla(i ,j)	-- Slack available for product 'i' at jth stage
EPT(i ,j)	-- Earliest pickup time of product i at jth stage
EDT(i ,j)	-- Earliest delivery time of product i at jth stage
LPT(i ,j)	-- Latest pickup time of product i at jth stage
LDT(i ,j)	-- Latest pickup time of product i at jth stage
APT(i ,j)	-- Actual pickup time of product i at jth stage
ADT(i ,j)	-- Actual delivery time of product i at jth stage
Pshift	-- Shift due to the insertion of the pickup node (Local variable).
Dshift	-- Shift due to the insertion of the delivery node(Local variable).

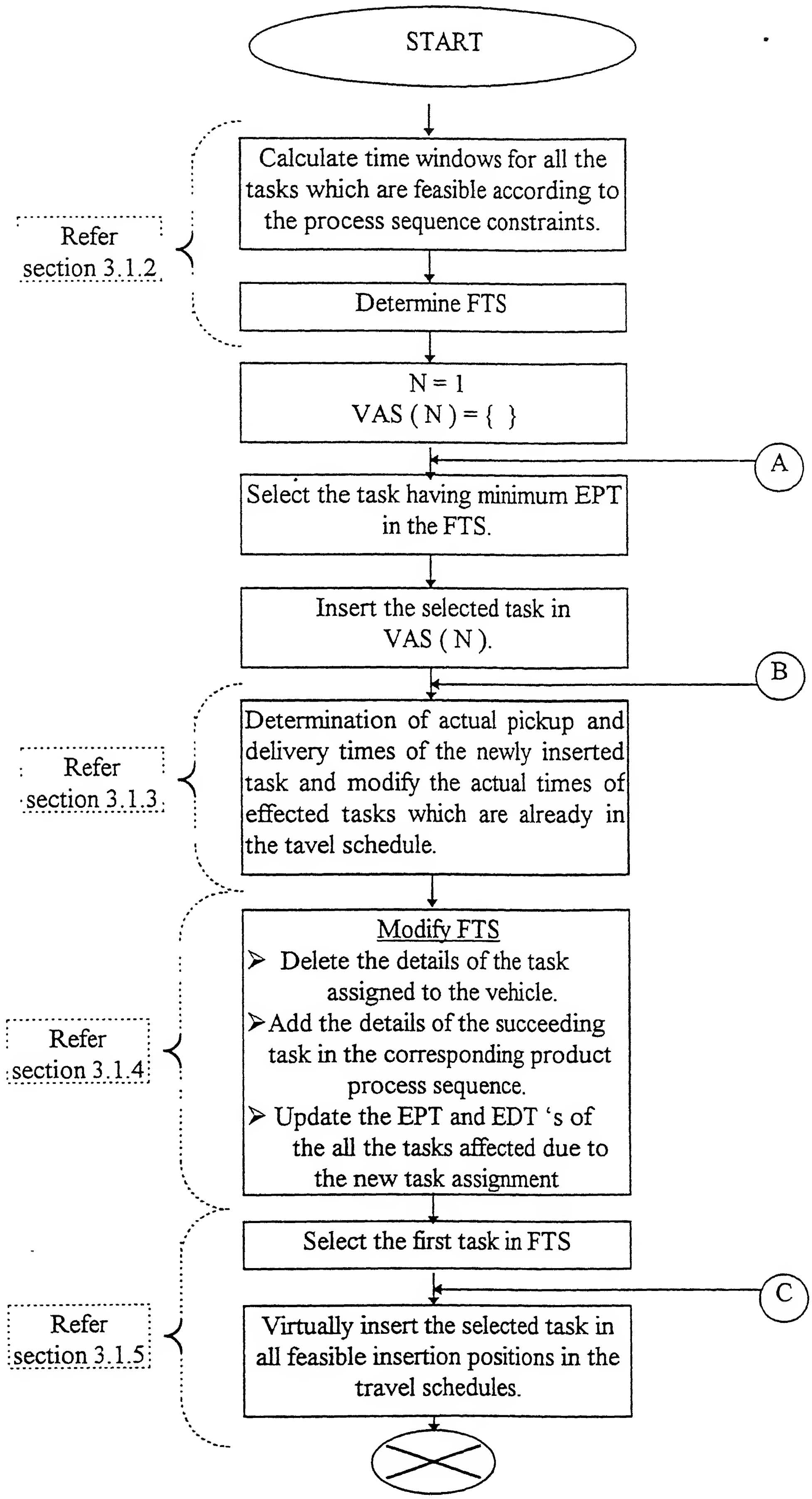


Fig 3.1.1 : Flow chart for Sequential Insertion Algorithm

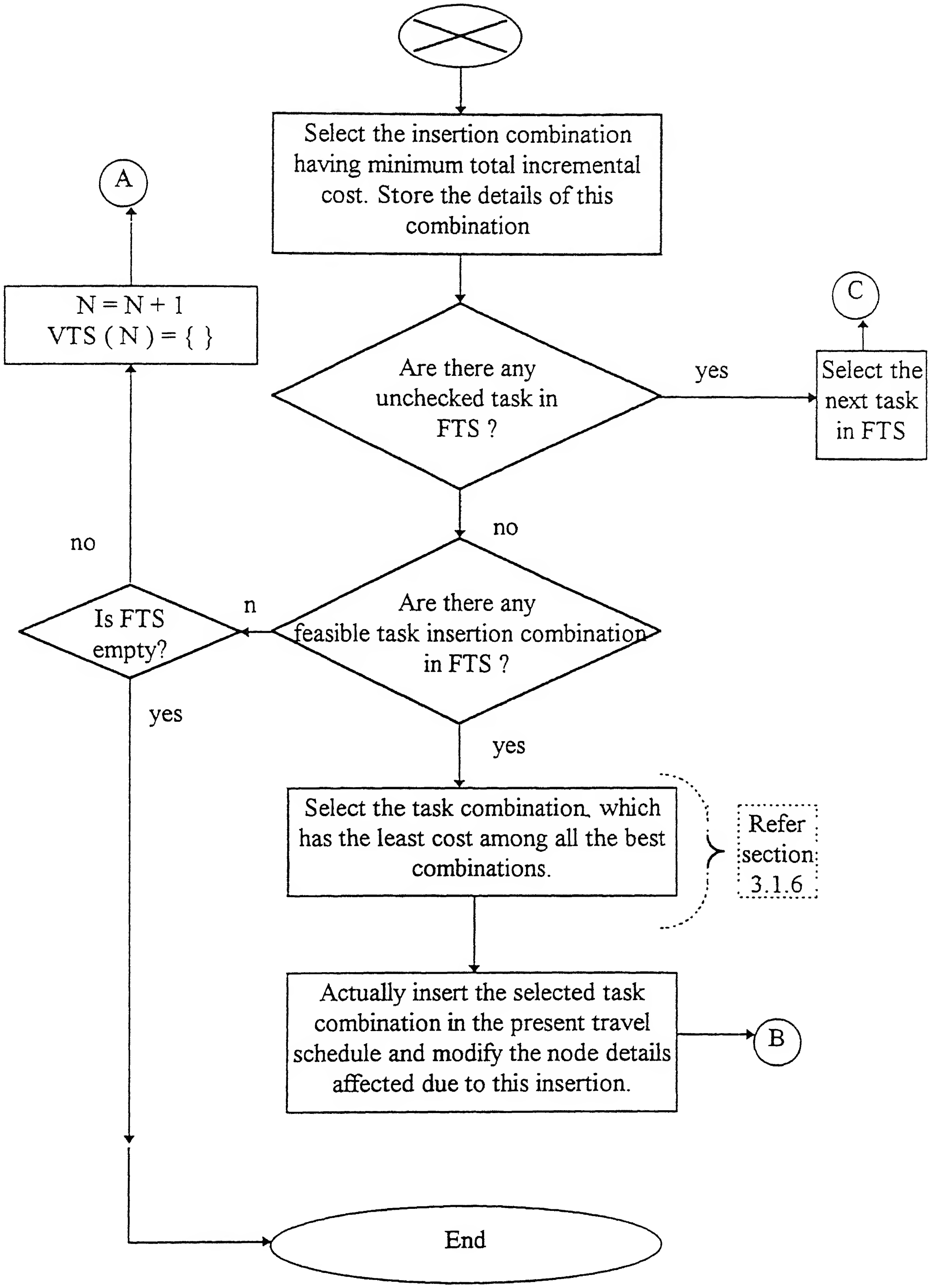


Fig 3.1.1 : Flow chart for Sequential Insertion Algorithm (Continue)

The sequential insertion algorithm (SEQIA) first calculates the time windows of all the sequence feasible tasks (Refer section 3.1.2). Then it stores the details of these tasks in FTS. SEQIA start by indexing the task having minimum EPT (Earliest Pickup Time) in FTS to the first vehicle. Then it determines the actual times of the inserted task.(Refer section 3.1.3). Then updates the FTS, i.e., it deletes details of the task assigned to the vehicle and add the details of the succeeding task in the corresponding product sequence (Refer section 3.1.4) Then it processes each task in FTS. Here, each task has a pickup and delivery node[1] Now, it starts checking the feasibility of insertion of each task in all possible insertion positions in the present travel schedule in a systematic manner (more details in Section 3.1.5). Among all the feasible insertion combinations it stores the details of the best combination (i.e., having minimum total incremental cost) for each task. Then it selects the task combination, which has the least incremental cost among all the best combinations stored in the previous step(Refer section 3.1.6). This task combination is actually assigned to the corresponding vehicle travel schedule. The algorithm calculates the actual time of occurrence of the newly assigned pickup and delivery tasks and the maximum shift possible for those tasks. It also modify the actual time and maximum shift possible values for all the nodes which are effected in the travel schedule of that vehicle (more details in Section 3.1.2.) due to the newly assigned task. It also updates the FTS (more details in section 3.1.3). SEQIA repeats this process until all the feasible task insertions are completed for that vehicle. If there is no feasible task to insert to that vehicle, then it select next vehicle to schedule and repeat the procedure till all the tasks to be done are completed (i.e., till FTS becomes empty).

---

[1] Here, a node represents either a pickup or delivery tasks in the schedule .

3.1.2. Determination of FTS :

This section describe the detailed calculations involved in the determination of FTS. At the beginning, the algorithm determines the time windows for each of the first stage tasks of all the products as follows:

$$EPT(i, 1) = Prt(i, 1);$$

$$LPT(i, 1) = EPT(i, 1) + Sla(i, 1);$$

$$EDT(i, 1) = EPT(i, 1) + Trt(P_{i1}, D_{i1});$$

$$LDT(i, 1) = EDT(i, 1) + Sla(i, 1);$$

3.1.3 Determination and modification of actual pickup and delivery times:

This section describes the calculation and modification procedure of actual pickup and delivery times of tasks. The calculation procedure can be better explained through an example. Consider a case as shown in figure 3.1.2.

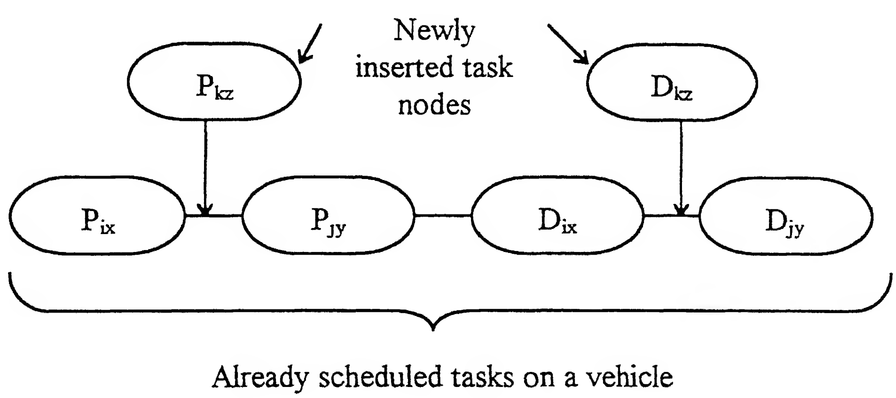


Fig 3.1.2:Illustration of determination and modification of actual pickup and delivery times

Here, P_{ix}, D_{ix} represent the pickup and delivery (p-d) nodes of i^{th} product batch at x^{th} stage . Similarly, P_{jy}, D_{jy}, P_{kz} & D_{kz} also represent the p-d nodes. Figure 3.1.2 shows that x^{th} and y^{th}

Stage tasks of i^{th} and j^{th} products respectively, are already scheduled on a vehicle. A new z^{th} stage task of k^{th} product is decided to be inserted in the combination as shown in figure 3.1.2. The algorithm first calculates the actual pickup time and the shift due to the insertion of the pickup node and then it will calculate the actual delivery time and modify the actual completion times of the tasks following these nodes by calculating the shift due to the insertion of delivery node as follows:

- <u>Calculation of the actual pickup and delivery times of the newly inserted task and shifts due to the insertion :</u>

$$APT(k, z) = \max.\{ EPT(k, z), (APT(i, x) + Trt(P_{ix}, P_{kz})) \}$$

$$Pshift = \underbrace{\max.\{0, (EPT(k, z) - (APT(i, x) + Trt(P_{ix}, P_{kz})))\}}_{\text{Waiting time}} + Trt(P_{kz}, P_{jy}) - Trt(P_{ix}, P_{kz})$$

$$ADT(k, z) = ADT(i, x) + Pshift + Trt(D_{ix}, D_{kz})$$

$$Dshift = Trt(D_{ix}, D_{kz}) + Trt(D_{kz}, D_{jy}) - Trt(D_{ix}, D_{jy})$$

- <u>Modification of actual times of the tasks which were effected by the newly inserted task :</u>

Due to the insertion of the new task, the tasks which are in between the newly assigned pickup node and delivery node are shifted by the amount Pshift. The tasks following the delivery node get shifted by the amount (Pshift+Dshift).

In the example shown in figure 3.1.2 APT(j, y) and ADT (i, x) get shifted by Pshift and ADT (j, y) get shifted by (Pshift + Dshift).

3.1.4 Modification of FTS :

This section describe the detailed calculations involved in the modification of FTS. Whenever a task is assigned to a travel schedule, the data in FTS is to be modified accordingly as follows:

- Calculates the time windows for the task succeeding the assigned task in its process sequence, as follows:

$$EPT\,(i, k) = ADT\,(i, k\text{-}1) + Trt\,(D_{ik\text{-}1}, P_{ik});$$

$$LPT\,(i, k) = EPT\,(i, k) + Sla\,(i, k);$$

$$EDT\,(i, k) = EPT\,(i, k) + Trt\,(P_{ik}, D_{ik});$$

$$LDT\,(i, k) = EDT\,(i, k) + Sla\,(i, k);$$

- Stores the pickup and delivery process centre numbers of the succeeding task.

- Deletes the information about the assigned task and add the information of the succeeding task.

Due to the newly inserted task, some of the tasks already assigned to the vehicle may be shifted. So, the algorithm modifies the EPT 's and EDT 's of all the effected tasks in FTS. The calculation procedure is better illustrated through the example discussed in the previous section (refer figure 3.1.2). As explained in the previous section, the algorithm first calculates the shift due to the pickup node insertion (i.e., Pshift) at the specified position. Due to this shift, completion times of all the succeeding nodes are shifted by this amount (i.e., for P_{jy}, D_{ix} & D_{jy}).

This 'Pshift' is added to the EPT 's and EDT 's of the effected product tasks (here i^{th} and j^{th} product tasks) in FTS. Then similarly, the algorithm calculates the shift due to delivery node insertion. This 'Dshift' is also added to the EPT 's and EDT 's of the effected product tasks (here j^{th} product task) in FTS.

3.1.5 Feasibility checking and virtual insertion procedure :

In the section 3.1.1, it was mentioned that the algorithm finds out the best feasible combination for each task in FTS, by inserting p-d nodes virtually in all feasible insertion positions. This section will describe the detailed procedure, for checking all the feasible combinations for a task selected from FTS in the travel schedule of a vehicle.

After selecting a task from FTS, the algorithm starts searching from the last node of the travel schedule of the vehicle, for the delivery node of the preceding task, in the process sequence of the selected task. If the preceding task was not assigned to that vehicle, or if the task to be assigned is not having any preceding task (i.e., it is the first task of the product), then the algorithm starts insertion after the first pickup node of the travel schedule. Otherwise, it starts insertion after the delivery node of that preceding task so that process sequence requirement is satisfied.

The insertion procedure can be better illustrated through an example. Consider a travel schedule shown in Figure. 3.1.3. Let the new task to be inserted is (P_{ix+1} , D_{ix+1}). Three product tasks at different stages are already scheduled on the vehicle. As described above the algorithm first searches for the first insertion position and it starts insertion after the D_{ix} node as shown in figure.3.1.3. to satisfy the process sequence constraint. Before inserting the pickup node it will check the following :

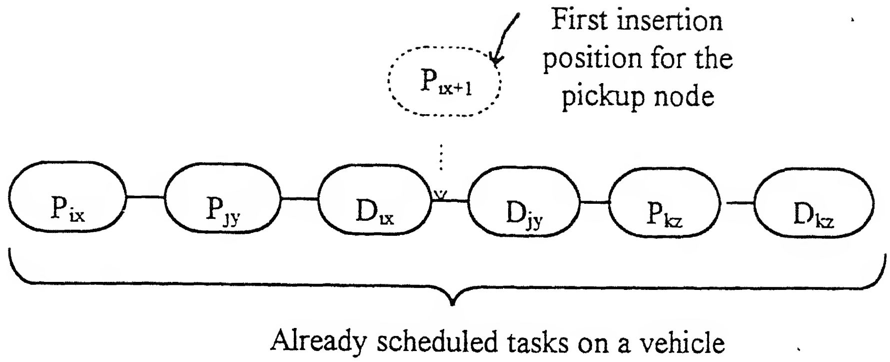


Fig 3.1.3: Illustration of the first insertion position for the pickup node

- First it will check the vehicle capacity constraint, by virtually adding increase in load due to the p-node insertion at the specified position. If the total load at this position does not exceed the vehicle capacity, then it further checks the time feasibility constraint, as described in the next point. Otherwise the insertion position is moved to the next position (i.e., after the node D_{jy}).

- To check the time feasibility constraint, the algorithm first find out the minimum shift permissible, by finding the least shift allowed among all the nodes following the insertion position. Then it calculates the actual shift to be done for that insertion. If this shift is less that or equal to the permissible shift, then the pickup node insertion at that position is feasible. Otherwise, the insertion position is moved to the next one.

After deciding the first insertion position of the pickup node, the algorithm starts checking the insertion of delivery node after the pickup node as shown in the figure.3.1.4. Similar to the procedure explained for pickup node insertion, the algorithm checks the time feasibility of the delivery node insertion, by comparing the permissible shift of the nodes following the insertion position of the delivery node. If it is feasible, the algorithm calculates and stores the total incremental cost (Section 3.1.6) involved in this combination and also stores the details of the combination.

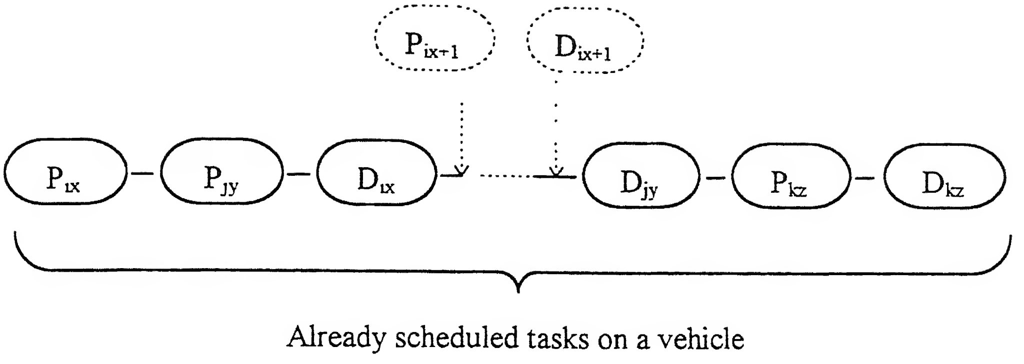


Fig 3.1.4 Illustration of the first insertion position for the delivery node

Then move the delivery task to the next insertion and repeat the procedure. If it is not feasible, it will simply move the delivery node to the next insertion position and repeat the procedure Like this the algorithm stores the details of all the feasible combinations, till the delivery task reaches the end of the schedule as shown in the figure.3.1.5.

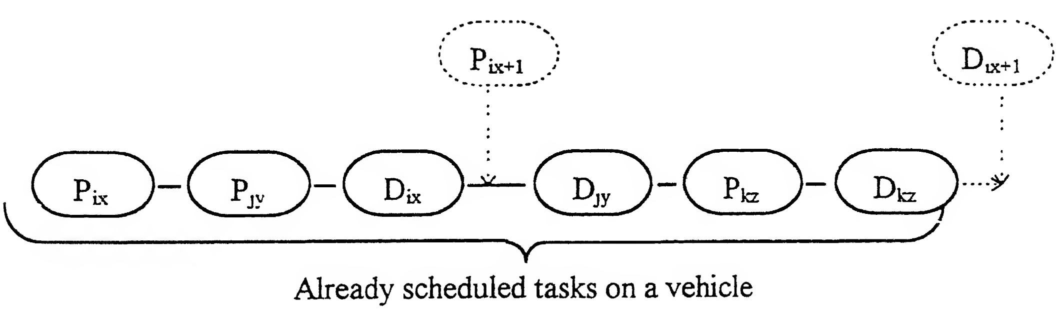


Fig 3.1.5 : Illustration of last insertion position for the delivery node

After finding all the feasible insertion combinations by keeping pickup node at first insertion position, the algorithm moves the pickup node to the next insertion position and finds out all the feasible insertion combinations as explained above. Repeat this procedure till the pickup node reaches the last insertion place. The last insertion combination is shown in the figure 3.1.6.

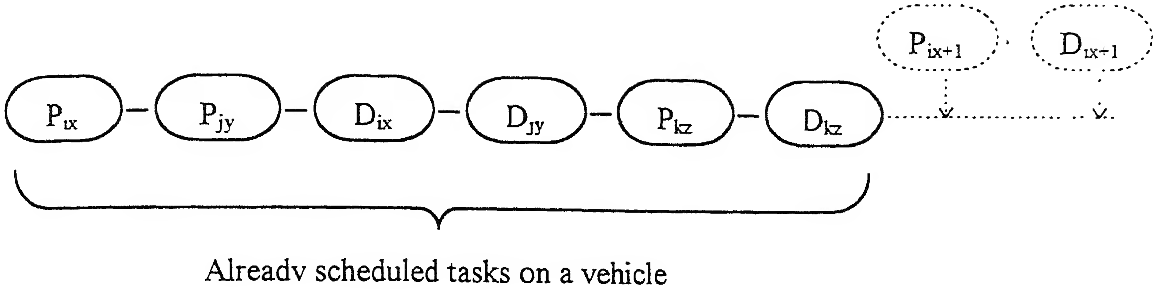


Fig 3.1.6 : Illustration of the last insertion combination of a p-d task

After checking all the combinations of that task , the algorithm selects the combination having minimum total incremental cost. It retains the details of that combination and the cost involved in that. Then it deletes the other combination details. After this the algorithm selects the next task in FTS and repeat the same procedure.

3.1.6 The optimisation procedure :

Inorder to select the best feasible insertion combination, among all the feasible insertion combinations of a task into the travel-schedules of the available vehicles, we use an objective function designed to evaluate the incremental cost involved in each insertion.

This cost is taken to be, the sum of the tangible and intangible costs involved in an insertion. Here, operating cost involved in travelling is considered as tangible cost. On the other hand to control the utilisation of the slack available for each product, slack utilisation cost and vehicle waiting time cost are considered as intangible costs.

The algorithm computes the total incremental cost for each feasible insertion combination. This computation procedure can be illustrated through the following example. Consider a feasible insertion combination as shown in figure3.1.7. in a vehicle travel schedule.

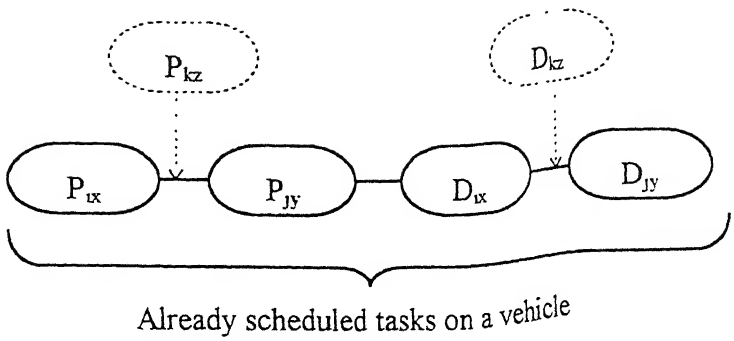


Fig 3.1.7 Illustration of the incremental cost calculation

The total incremental cost involved in the insertion is calculated as follows.

Total cost = Total increase in travel cost + vehicle waiting cost + slack utilisation cost

Total increase in travel cost = Increase in travel cost due to pickup node insertion

+ Increase in travel cost due to delivery node insertion

Increase in travel cost due to p-node insertion is given by

$$Ctr * [\ Dist\ (P_{ix}, P_{kz}) + Dist\ (P_{kz}, P_{Jy}) - Dist\ (P_{ix}, P_{Jy})\ ]$$

Increase in travel cost due to d-node insertion is given by

$$Ctr * [\ Dist\ (D_{ix}, D_{kz}) + Dist\ (D_{kz}, D_{jy}) - Dist\ (D_{ix}, D_{jy})\ ]$$

$$\text{Vehicle waiting cost} = Cwt * [\ \max.\ \{\ 0, [\ EPT_{kz} - (\ APT_{ix} + Trt\ (P_{ix}, P_{kz}\ )\ )\ ]\ \}\ ]$$

$$\text{Slack utilisation cost} = Csl * [\ \sum_{i=1}^{n_p} Slu\ (\ i\ )\ ]$$

Here, the algorithm calculates all the products slack utilised (Slu) in a feasible insertion into the travel schedule to calculate the slack utilisation cost. This calculation procedure can be better illustrated through the example shown in figure 3.1.7.

It first calculates the slack utilised for the product, whose pickup node is virtually inserted (i.e., here, k^{th} product) and the products whose tasks are following the pickup node insertion position (i.e., here i^{th} and j^{th} products) as follows:

Product k^{th} slack utilised $PSlu(k) = APT_{kz} - EPT_{kz}$;

Products i and j's slack utilised $PSlu(i)$ & $PSlu(j)$

$= \max.\{0, (EPT_{kz} - (APT_{ix} + Trt(P_{ix}, P_{kz}))\} + Trt(P_{ix}, P_{kz}) + Trt(P_{kz}, P_{jy}) - Trt(P_{ix}, P_{jy})$;

For all other products PSlu is assigned to be zero.

Then the slack utilised in the delivery node insertion is calculated, similarly as described above for pickup node insertion as follows:

Product k^{th} slack utilised $DSlu(k) = (ADT_{kz} - EDT_{kz}) - PSlu(k)$;

Product j^{th} slack utilised $DSlu(k) = Trt(D_{ix}, D_{kz}) + Trt(D_{kz}, D_{jy}) - Trt(D_{ix}, D_{jy})$;

For all other products the DSlu is assigned to be zero.

The total slack time utilised for all products (Slu) in the insertion is sum of PSlu and DSlu. Here, the performance of the algorithm depends upon the values of C_{st} and Cwt . According to these values the slack utilisation is controlled. The optimum values of these parameters are problem dependent. So, here the algorithm is repeated for various values of C_{st} and Cwt ,within a range. It stores the details of the schedules having minimum total cost, including the fixed cost associated per vehicle.

3.2 PARALLEL INSERTION ALGORITHM (PARIA):

3.2.1 Overview of the algorithm :

The major difference between PARIA and SEQIA is that, in PARIA inserting on a new vehicle is started whenever a task has no feasible insertion combination on all the

existing vehicles. Where as in SEQIA, insertion on a new vehicle is started, only when there are no feasible tasks to insert on all the existing vehicles. In PARIA a task is inserted on all the available vehicle travel schedules, in all feasible combinations and select the best one among those is selected. Similar to the SEQIA, an insertion combination of a task into a travel schedule is feasible only if it does not lead to the violation of vehicle capacity, time feasibility and process sequence constraints, for the newly assigned task and for all other tasks already scheduled on the vehicles. The optimisation step deals with minimising the incremental cost due to insertion of new task into a vehicle travel schedule.

Step by step procedure of the algorithm is illustrated through flow chart in figure 3.2.1. The parallel insertion algorithm (PARIA), first calculates the time windows to all the sequence feasible tasks. Then it stores the details of these tasks in FTS. Similar to SEQIA, this algorithm also starts by indexing the task having minimum EPT in FTS to the first vehicle. Then update the FTS and starts insertion procedure for the new task assignment on the first vehicle. This process of insertion on the first vehicle is same as in SEQIA, till for a sequence feasible task no feasible insertion combination is on this vehicle i.e., task is infeasible on this vehicle. Then the algorithm insert this first such task on the second vehicle. Now, the algorithm starts checking for the feasible insertion combinations of all the tasks in FTS, on both the vehicle travel schedules. It selects the task combination which has minimum incremental cost, among all the feasible insertion combinations. The virtual insertion procedure in PARIA is similar to SEQIA, except that it has to check for the best feasible insertion on all the available vehicle travel schedules. The optimisation procedure is also same as in SEQIA.

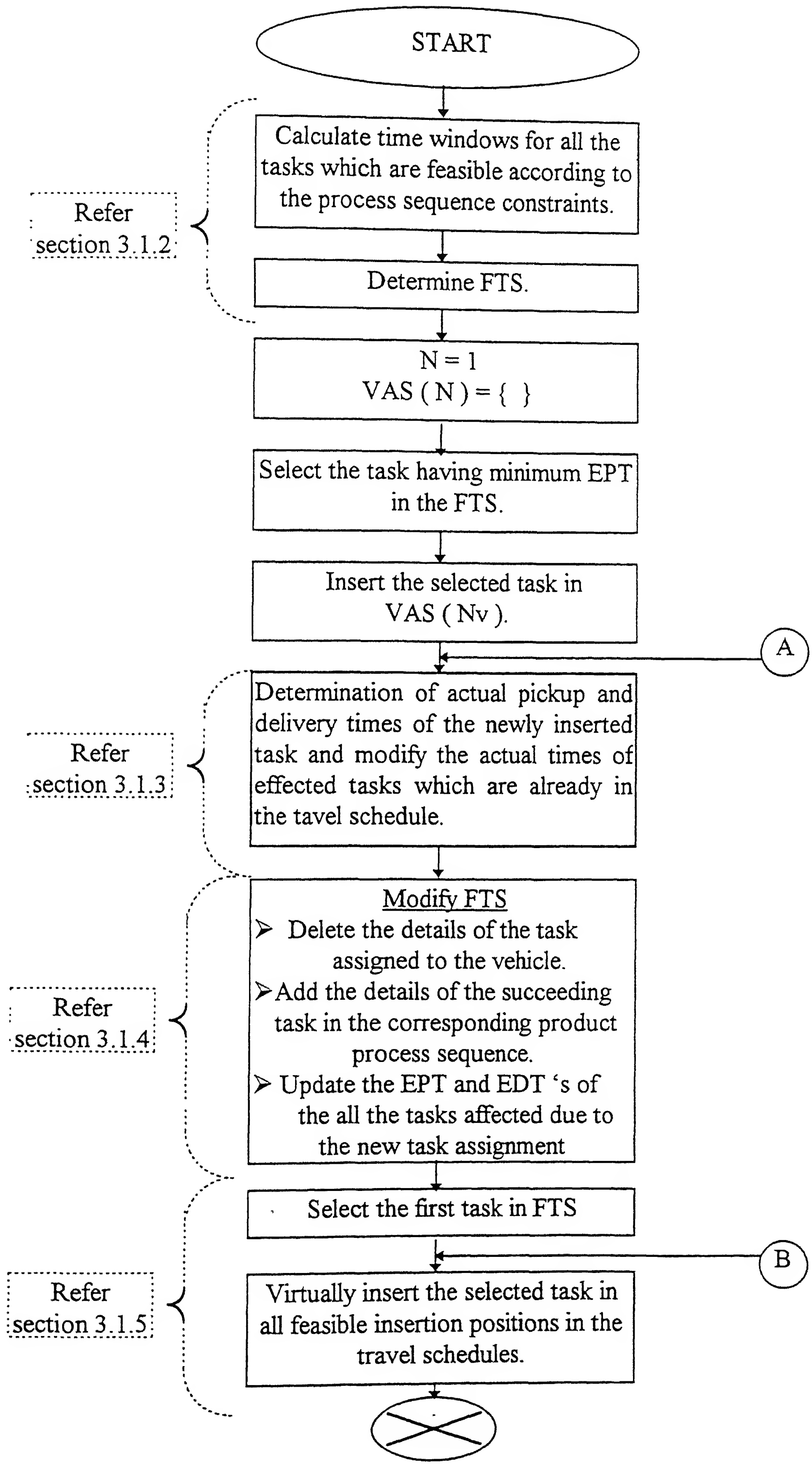


Fig 3.2.1 : Flow chart for Parallel Insertion Algorithm

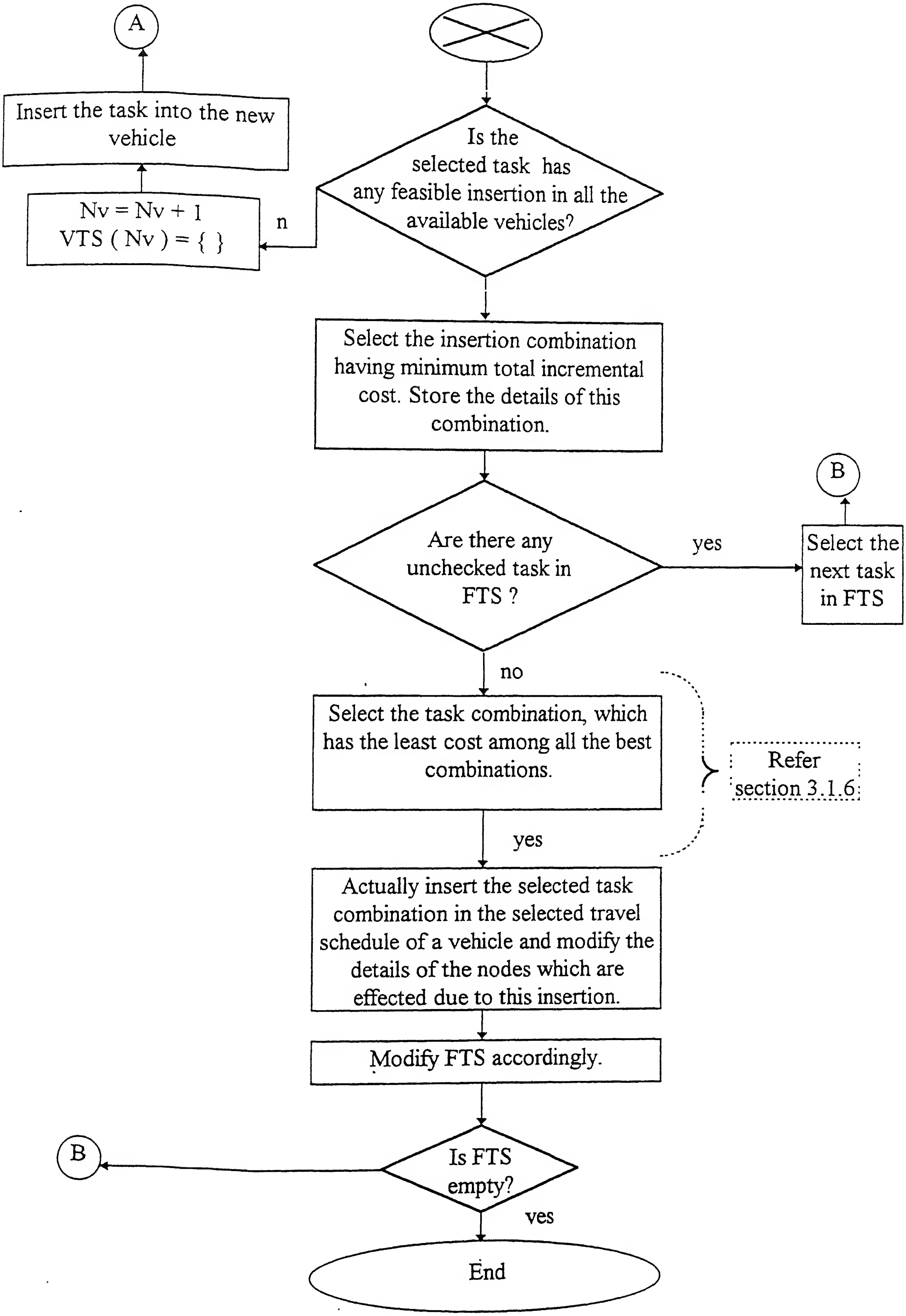


Fig 3.2.1 Flow chart for Parallel Insertion Algorithm *(Continue)*

3.3 HEURISTICS WITH DESPATCH RULES :

In both the heuristic approaches, the task selection is done by optimising the incremental cost with the various values of slack cost and vehicle waiting cost. This approach is quite time consuming and computationally intensive. In this section we describe four heuristics based on simple despatching rules for task selection. These rules are used for both sequential and parallel insertion heuristics.

3.3.1 Earliest due date procedure (EDD) :

Jackson [1955] proved that scheduling according to EDD rule, will minimise the maximum lateness. As we are looking for a travel schedule in which all tasks are to be completed within their due-dates, selection of the task for insertion using EDD is likely to be effective. According to this rule, the algorithm selects the task which has the minimum latest delivery time. After selecting the task, the algorithm virtually inserts the pickup and delivery nodes of this task in all feasible insertion positions. If there is no feasible insertion for this task, then it selects the next best according to this rule. Among all the virtual insertion combinations of the selected task, the algorithm selects the combination which has the least incremental travelling cost.

3.3.2 First come first serve (FCFS) :

According to this rule the algorithm selects the task which has minimum earliest pickup time and virtually inserts the pickup and delivery nodes of this task in all the feasible insertion positions. If there is no feasible insertion for this task, then it selects the next best task

according to FCFS rule. Among all the virtual insertion combinations of the selected task it chooses the combination which has the minimum incremental travelling cost. The motivation behind this rule is that, the vehicle is loaded with a task as soon as the task is available. This is likely to result in reduced waiting times for the vehicles.

3.3.3 Minimum slack per operation (MINSL):

According to this rule, the algorithm selects the task of a product which has minimum slack available per operation at that stage. After the selection, it virtually inserts the pickup and delivery nodes of this task in all the feasible insertion positions. Among all the virtual insertion combinations of this task, it chooses the combination which has the least incremental cost This rule will effectively give priority to those tasks which have smaller slack available.

3.3.4 Systematic slack costing (SYSSL) :

In this heuristic penalty is imposed for over utilisation of the slack per operation available for a task. For a task insertion combination, difference between actual slack utilised and the slack available per operation for that product are computed. Slack over utilisation cost is computed as the product of penalty per unit slack over utilisation and the excess slack utilised. We have selected the penalty for over utilisation of the slack equal to the travel cost per period.

3.4 Restricted number of vehicles at the depot :

The algorithms discussed in the previous sections deal with the problems for which the violation of due-dates (i.e., hard time windows) are not allowed and also without any

restriction on the number of vehicles available at the depot. But, in the practical scenario, the number of vehicles available at the depot are restricted. So, we made an attempt to observe the performance of these algorithms in case of restricted number of vehicles at the depot In this case the violation of due-dates are allowed, when the available number of vehicles are not sufficient to meet all the transportation requests.

To control the violation of the due-dates of the products, a penalty cost per period violation of due-date is included in the calculation of incremental cost in the algorithms. Whenever, a new task insertion violates the due-date of that task or the due-dates of the tasks which are already scheduled to that vehicle: the algorithm calculates the penalty cost involved in that insertion and add this to the incremental cost(Refer section 3.4.1).

3.4.1 Calculation of penalty cost :

The calculation procedure can be better illustrated through an example. Consider a sequence feasible insertion combination as shown in figure 3.4.1.

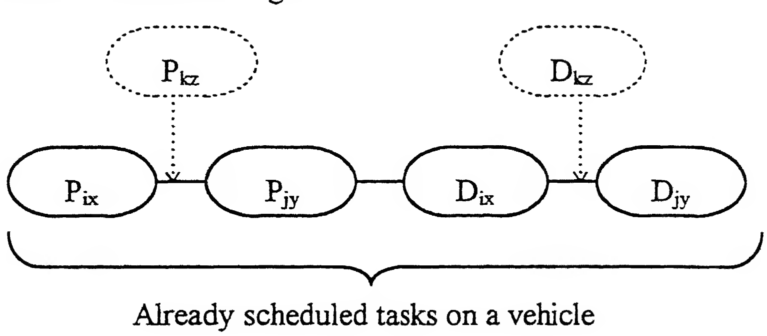


Fig 3.4.1: Illustration of calculation procedure of penalty cost

If this task insertion combination violates the due-dates. The algorithm calculates the penalty cost as follows:

- if ADT (k, z) > LDT (k, z)

  then

  tpen (k, z) = ADT (k, z) - LDT (k, z);

- for the tasks which are between the pickup and delivery node insertion positions (i.e., Pjy and Dix), the algorithm compares the shift due to the pickup node insertion and the shift available for those tasks. If the available shift is less than the actual shift due to the pickup node (Pshift), then the algorithm calculates the difference between these and add this to the tpen(k, z).

- for the tasks which are following the delivery node, the algorithm compares the shift due to both the pickup and delivery nodes insertion and the shift available for those tasks. If the available shift is less than the actual shift due to both the pickup and delivery node (Pshift+Dshift) insertion, then the algorithm calculates the difference between these and add this to tpen(k, z).

Penalty cost = Cpen*tpen(k, z);

CHAPTER 4

COMPUTATIONAL STUDY

In this chapter we will describe the details of computational study conducted for both the sequential and Parallel Insertion algorithms and the dispatch rules. In the section 4.1 we will describe the design of the test problems, which are used for the performance analysis of the heuristics. In section 4.2 we will describe the performance analysis of all the algorithms discussed in the previous chapter. In section 4.3 we will study time complexity of algorithms In section 4.4 we will discuss the performance of heuristics, when number of vehicles is fixed.

4.1 Design of test problems :

A variety of test problems were generated randomly. In this section we will describe the details of the generation of these test problems. In the design of test problems several factors that can effect the behaviour of heuristics were considered. We divided these factors into following two categories;

I . Problem process data :

(1) Processing centres utilisation,

(2) Slack available for the products,

(3) Processing time to travelling time ratio.

II . Problem size :

(1) Number of products,

(2) Number of operations per product,

(3) Number of processing centres.

To generate the problems of required class, we programmed a problem generator. The complete details of the working of this generator are illustrated through flow chart as shown in figure 4.1. In this, we used a standard routine available to generate the pseudo random number. The first seed to this random number generator is given through standard input.

4.1.1 Process data variation:

In this section we will describe the procedure followed to generate the problems by varying the process data of the problem. We generated twenty seven problems, by keeping the factors-- process centre utilisation, slack time available and processing time to travel time ratio at three levels. The actual combinations used are described in Appendix-A. The details of these factor levels are discussed below:

i) <u>Process centre utilisation</u>:

For this factor the following three levels are considered.

Level -1 : *Equal* utilisation of all the processing centres.

For example if there are four process centres. Each process centre has 25% utilisation.

Level -2 : A combination of *high, medium and low* process centres utilisation.

For example if there are six process centres. Two of them will have high percentage utilisation (say 35%, 30%), the other two will have medium percentage utilisation (say 10%, 15%) and the last two problems will have low level of utilisation (say 3%, 7%).

Level -3 : A combination of high and low process centres utilisation.

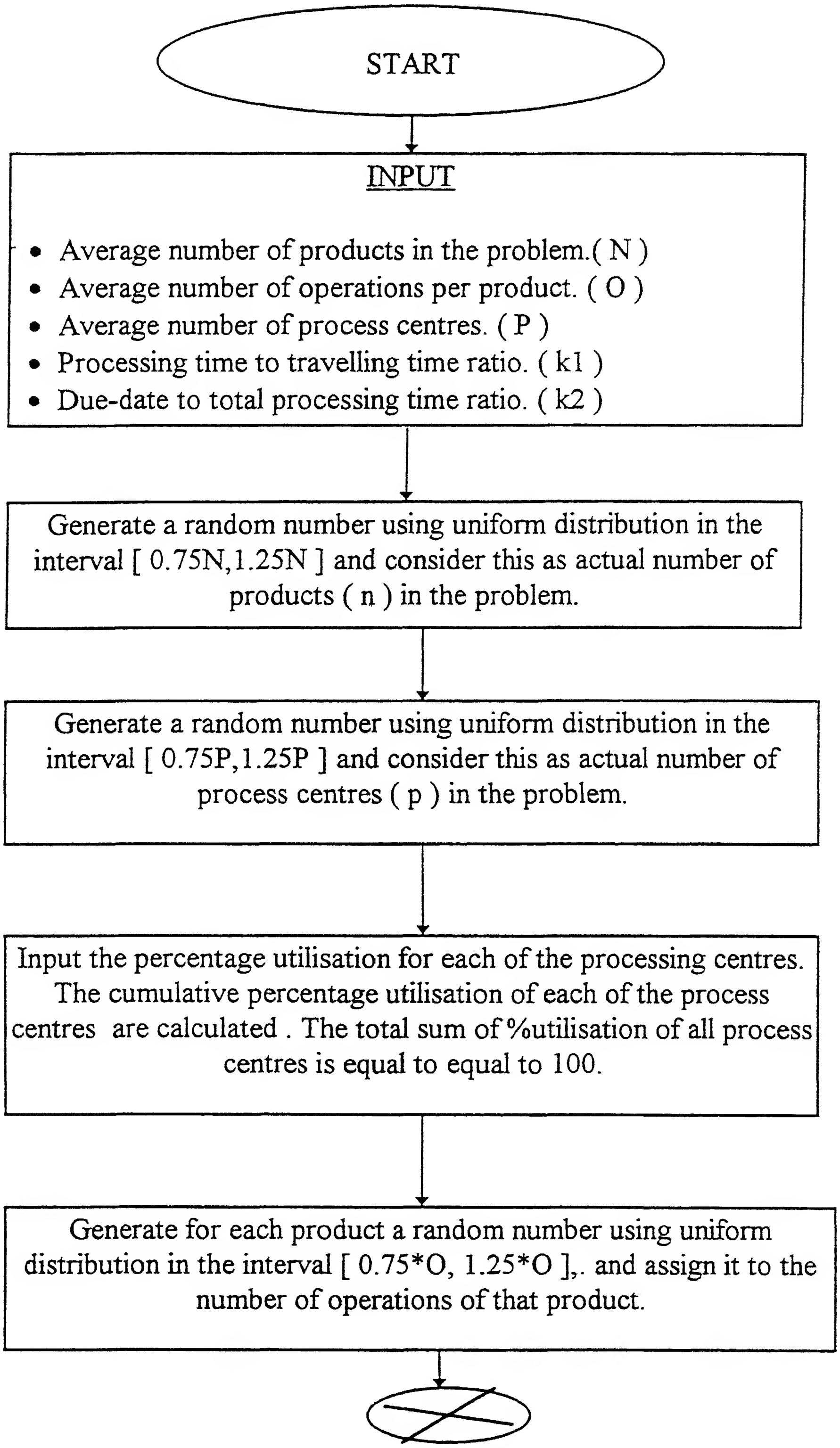


Fig 4.1 Flow chart for the problem generator

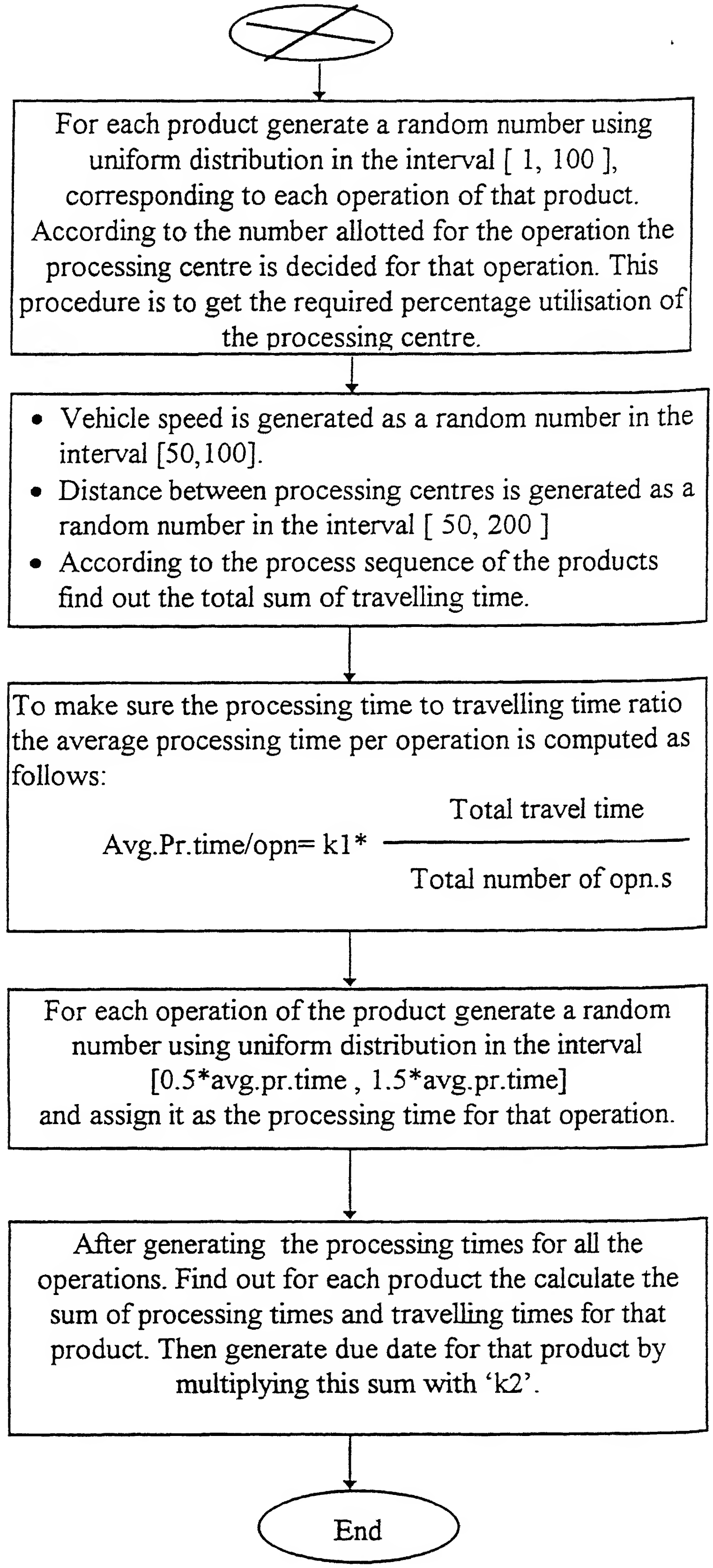


Fig 4.1 Flow chart for the problem generator(Continue)

For example if there are four process centres.Two of them have high percentage utilisation (say 42%, 43%), the other two have low level utilisation (say 8%, 7%).

ii)Slack time available per product:

For this factor the following three levels are considered.

Level-1 : High

Slack time available for a product is equal to the total process time of that product.

Level-2 : Medium

Slack time available for a product is equal to 50% of the total processing time of that product.

Level-3 : Low

Slack time available for a product is equal to the 10% of the total processing time of that product.

iii) Processing time to travelling time ratio :

For this factor the following three levels are considered.

Level-1 : High

At this level ratio is kept at 4. It means that comparatively the travelling time involved in the problem is 25% of the processing time.

Level-2 : Medium

At this level ratio is kept at 2. It means that comparatively the travelling time involved in the problem is 50% of the processing time.

Level-3 : Low

At this level ratio is kept at 1. It means that comparatively the travelling time involved in the problem is 100% of the processing time.

4.1.2 Problem size variation:

In this section we will describe the procedure followed to generate the problems by varying the size of the problems. We generated forty problems, by keeping the factors-- number of products, number of operations per product and number of processing centres at two levels The actual combinations used are described in Appendix-B. For each combination five problems are generated with different process data The details of these factor levels are discussed below:

i) Number of products:

For this factor the following two levels are considered.

Level-1 : High

At this level number of the products for a problem is generated randomly in the range 10-18.

Level-2 : Low

At this level number of the products for a problem is generated randomly in the range 4-10.

ii) Number of operations per product:

For this factor the following two levels are considered.

Level-1 : High

At this level number of the operations per product for a problem is generated randomly in the range 10-20.

Level-2 : Low

At this level number of the operations per product for a problem is generated randomly in the range 5-10.

iii) Number of process centres:

For this factor the following two levels are considered.

Level-1 : High

At this level number of process centres for a problem is generated randomly in the range 10-15.

Level-2 : Low

At this level number of the process centres for a problem is generated randomly in the range 5-10.

The detailed computational study is explained in the next section.

4.2 Performance analysis :

In this section we will describe the computational performance of the sequential and parallel heuristics discussed in the previous chapter and the performance of the heuristics with despatch rules, which were discussed in section 3.3. We programmed all these algorithms in C_language and performed computational test on all the problem classes described in the previous section. All the computational experiments are done on HP9000 platform using UNIX operating system.

To compare the performance of the algorithms and despatch rules among them self, we considered the following two measures :

1) Average % deviation of the total travelling cost of an algorithm from the best solution among all the heuristics.
2) Average number of vehicles used.

Graphs for different problem sets were plotted by taking the average percentage deviation of total travelling cost on the y-axis and the despatch rule used on the x-axis (Fig 4.2 & Fig 4 4) and also by taking the average number of vehicles used on y-axis and the despatch rule used on the x-axis (Fig 4.3 & Fig 4.5). The details of the behaviour of these algorithms for different problem sets are discussed below.

4.2.1 Processing centre utilisation :

By keeping the processing centre utilisation at three different combinations, we performed computational test for nine problems corresponding to each of the combination of slack available and processing time to travel time ratio. Following is observed

(1) *Average percentage deviation of total travelling cost from the best one* :

According to this measure:

a) at the best values of slack cost and vehicle waiting cost the Parallel Insertion algorithm is performing better than the Sequential Insertion algorithm for all the three combinations of processing centres utilisation.(Refer Fig 4.2.1)

b) Among the despatch rules applied to Sequential Insertion algorithm the Systematic Slack Costing rule is performing better than other three rules in case of the even and also High-Medium-Low utilisation combination of the processing centres. But for

the case of high-low utilisation combination of the processing centres First Come First Serve is performing better than others. .(Refer Fig 4.2.1)

c) Among the despatch rules applied to Parallel Insertion algorithm the first come first serve rule is performing better than the other rules for all the three combinations.

d)Among all the despatch rules applied for both the algorithms, the first come first serve applied to Parallel Insertion procedure is performing better (Refer Fig 4.2.1 & Table 4.3).

(2) *Average number of vehicles used* :

According to this measure:

(a) Both the sequential and Parallel Insertion algorithms are performing almost equally at the best values of slack utilisation cost and vehicle waiting cost.

(b) For both the algorithms, the First Come First Serve rule is performing better than the other rules.

(c)The First Come First Serve rule applied to the Parallel Insertion case is performing better among all the despatch rules applied to both the algorithms for all the three combinations. (Refer Fig 4.3)

4.2.2 Processing time to travelling time ratio :

By keeping processing time to travelling time ratio at high, medium and low levels, the computational tests are performed for nine problems corresponding to each of the combination of process centre utilisation and slack available. The details of these results are described as follows.

(1) *Average percentage deviation of total travelling cost*:

According to this measure:

(a) For all the three levels of processing time to travelling time ratio, the parallel insertion algorithm is performing better than the Sequential Insertion algorithm at the best values of slack cost and vehicle waiting cost.

(b) Among the despatch rules applied to Sequential Insertion algorithm the first come first serve rule is performing better than the other three rules when this ratio is at high level, Systematic Slack Costing is performing better when this ratio is at medium level and Earliest Due Date rule is performing better when this ratio is at low level.

(c) Among all the despatch rules applied to the Parallel Insertion algorithms, the First Come First Serve rule is performing better than others at all the three levels.

(d)Among all the despatch rules applied for both the algorithms, the First Come First Serve applied to Parallel Insertion procedure is performing better (Refer Fig 4.2 & Table 4.3).

(2) *Average number of vehicles used* :

According to this measure

(a) Both the sequential and Parallel Insertion algorithms are performing almost equally at the best values of slack utilisation cost and vehicle waiting cost.

(b) For both the algorithms, the First Come First Serve rule is performing better than the other rules.

(c) The First Come First Serve rule applied to the Parallel Insertion case is performing better among all the despatch rules applied to both the algorithms at all three levels (Refer Fig 4.3 & Table 4.4).

4.2.3 Slack time available per product :

By keeping the slack time available per product at high, medium and low levels for all the products, the computational test are performed for nine problems corresponding to each of the combination of process centre utilisation and processing time to travel time ratio. The details of these results are described as follows.

(1) *Average percentage deviation of total travelling cost*:

According to this measure

(a) For all the three levels of slack time available per product, the Parallel Insertion algorithm is performing better than the Sequential Insertion algorithm at the best values of slack cost and vehicle waiting cost.

(b) Among the despatch rules applied to the Sequential Insertion algorithm the first come first serve rule is performing better than the other three rules when all the products in the problem have slack availability at high level or at low level, Systematic Slack Costing rule is performing better when all the products in the problem have slack availability at medium level.

(c) Among all the despatch rules applied to the Parallel Insertion algorithms, the First Come first Serve rule is performing better than others at all the three levels.

(d) Among all the despatch rules applied for both the algorithms, the First Come First Serve applied to Parallel Insertion procedure is performing better (Refer Fig 4.2 & Table 4.3).

(2) *Average number of vehicles used:*

According to this measure:

(a) Both the sequential and Parallel Insertion algorithms are performing almost equally at the best values of the slack utilisation cost and vehicle waiting cost, when the slack available for all products are at high and medium levels.

(b) In case the slack availability is at low level for all the products, the parallel insertion method is performing better than the sequential insertion.

(c) For both these algorithms when the slack available for all products are at high and medium levels, the despatch rule First Come First Serve is performing better than the other three rules.

(d) In case the slack availability is at low level, Earliest Due Date rule is performing better for both the algorithms.

(e) Among all the despatch rules applied to both the algorithms, the first come first serve rule applied to the Parallel Insertion algorithm is performing better, when the slack available for all products are at high and medium levels(Refer Fig 4.3 & Table 4.4). Similarly, the Earliest Due Date rule applied to the Parallel Insertion algorithm is performing better when the slack availability for all products are low.

4.2.4 Problem size factors:

As mentioned already, number of products, number of operations per product and number of departments are considered as three size parameters. By keeping each of these parameters at two levels, we generated six sets of problems. For each problem set we generated twenty problems. The details of these results are as follows :

(1) <u>Average percentage deviation of total travelling cost</u>:

According to this measure

(a) For all the six set of problems, the Parallel Insertion algorithm is performing better than the Sequential Insertion algorithm at the best values of slack cost and vehicle waiting cost.

(b) Among the despatch rules applied to the Sequential Insertion algorithms, Systematic Slack Costing rule is performing better than all other rules for all the six set of problems.

(c) Similarly, among all the despatch rules applied to the Parallel Insertion algorithms, the First Come First Serve rule is performing better than others except in the case of number of products are low, in this case Earliest Due Date despatch rule is performing better.

(d) Among all the despatch rules applied for both the algorithms, the first come first serve applied to Parallel Insertion procedure is performing better, when the number of products are at high level, number of operations per product are low and number of departments are low. The Systematic Slack Costing rule is performing better when the number of products are low, number of operations per product are high and number of departments are low (Refer Fig 4.4 & Table 4.5).

(2) Average number of vehicles used :

.....According to this measure

(a) Both the sequential and Parallel Insertion algorithms are performing almost equally for all the six sets of problems.

(b) Among all the despatch rules applied to both the algorithms, the first come first serve rule applied to the Parallel Insertion algorithm is performing better for all six type of problem sets (Refer Fig 4.4).

Summery of this analysis is shown in table 4.1 and table 4.2.

BEST PERFORMED DESPATCH RULES ON DIFFERENT PROBLEMS

(In terms of average%deviation of total travel cost from the best cost)

FACTOR	LEVEL	Best performed dispatching rule in case of sequ. insertion method	Best performed dispatching rule in case of parallel insertion method	Among all the dispatching rules for bothe the algorithm best performed one
PROCESS CENTRE UTILISATION	Even	SYSSL	FCFS	FCFS (Parl)
	High-Medium-low	SYSSL	FCFS	FCFS (Parl)
	High-Low	FCFS	FCFS	FCFS (Parl)
PROCESSING TIME TO TRAVELLING TIME RATIO	High	FCFS	FCFS	FCFS (Parl)
	Medium	FCFS	FCFS	FCFS (Parl)
	Low	SYSSL	FCFS	FCFS (Parl)
SLACK TIME AVAILABILITY	High	FCFS	FCFS	FCFS (Parl)
	Medium	SYSSL	FCFS	FCFS (Parl)
	Low	FCFS	FCFS	FCFS (Parl)
NUMBER OF PRODUCTS	High	SYSSL	FCFS	FCFS (Parl)
	Low	SYSSL	FCFS	SYSSL (Seq)
NUMBER OF OPERATIONS PER PRODUCT	High	SYSSL	FCFS	SYSSL(Seq)
	Low	SYSSL	FCFS	FCFS(Parl)
NUMBER OF PROCESSING CENTRES	High	SYSSL	FCFS	SYSSL(Seq)
	Low	SYSSL	FCFS	FCFS (Parl)

Table 4.1

BEST PERFORMED DESPATCH RULES ON DIFFERENT PROBLEMS
(In terms of average number of vehicles used)

FACTOR	LEVEL	Best performed dispatching rule in case of sequ. insertion method	Best performed dispatching rule in case of parallel insertion method	Among all the dispatching rules for bothe the algorithm best performed one
PROCESS CENTRE UTILISATION	Even	FCFS	FCFS	FCFS (Parl)
	High-Medium-low	FCFS	FCFS	FCFS (Parl)
	High-Low	FCFS	FCFS	FCFS (Parl)
PROCESSING TIME TO TRAVELLING TIME RATIO	High	FCFS	FCFS	FCFS (Parl)
	Medium	FCFS	FCFS	FCFS (Parl)
	Low	FCFS	FCFS	FCFS (Parl)
SLACK TIME AVAILABILITY	High	FCFS	FCFS	FCFS (Parl)
	Medium	FCFS	FCFS	FCFS (Parl)
	Low	FCFS	FCFS	FCFS (Parl)
NUMBER OF PRODUCTS	High	FCFS	FCFS	FCFS (Parl)
	Low	EDD	FCFS	FCFS (Parl)
NUMBER OF OPERATIONS PER PRODUCT	High	FCFS	FCFS	FCFS(Parl)
	Low	FCFS	FCFS	FCFS(Parl)
NUMBER OF PROCESSING CENTRES	High	FCFS	FCFS	FCFS(Parl)
	Low	FCFS	FCFS	FCFS (Parl)

Table 4.2

The following is observed :

%deviation of total travelling cost of a heuristic from the best one:

According to this measure:

- At the best values of slack cost and vehicle waiting cost Parallel Insertion algorithm is performing better than the Sequential Insertion algorithm, irrespective of the problem type.
- Among the despatch rules applied to the Sequential Insertion algorithm, Systematic Slack Costing rule is performing better inmost of the cases(67 %). In some of the cases First Come First Serve rule(33 %) is performing better.
- Among the despatch rules applied to the Parallel Insertion algorithm, First Come First Serve is performing better in all the cases.
- Among the despatch rules applied to both the Sequential and Parallel Insertion algorithms the First Come First Serve rule applied to Parallel Insertion algorithm is performing better except when the number of products are low, number of operations per product are high. and number of process centres are low. In these three cases Systematic Slack Utilisation applied to Sequential Insertion algorithm is performing better.

Average number of vehicles used:

According to this measure

- First Come First Serve despatch rule is performing better in both the Sequential and Parallel Insertion algorithms.
- First Come First Serve despatch rule applied to Parallel Insertion algorithm is performing

4.3 Time complexity of algorithms :

After testing the computational performance of the algorithms, we were interested to check the performance of the algorithms in terms of computational time taken for different sizes of the problems. The fig 4.6 illustrates the growth rate of computational time taken by these algorithms with respect to the number of pickup-delivery tasks in the problem. By observing the graph we can conclude that the growth rate of the time taken by the Parallel Insertion algorithm is more than that of Sequential Insertion algorithm. The worst case analysis of these algorithm shows that the Sequential Insertion algorithm has $O(n^3)$ and for Parallel Insertion algorithm $O(n^4)$ where 'n' is the total number of tasks in the problem.

4.4 Performance analysis in case of restricted number of vehicles available:

We performed a computational test for five problems and observed the behaviour of all the heuristics, by changing the weight of the penalty cost with respect to the travelling cost. Similar study is also comducted by changing the number of vehicles available at the depot. The computational results of an example problem are described in table 4.5,4.6,4.7 and 4.8. The observations made through these results are as follows :

- At the best values of slack cost and vehicle waiting cost Parallel Insertion algorithm is performing better than Sequential Insertion algorithm.
- Despatch rules applied to Sequential Insertion procedure are performing effectively when compared to the despatch rules applied to the Parallel Insertion procedure.

- Minimum slack per operation rule is performing better in case of Sequential Insertion method, in controlling the violation of due-dates. Where as the Systematic Slack Costing rule is performing better in reducing the travelling cost, but it is not able to control the violation of due-dates of the products.
- Earliest due date despatch rule is performing better in case of Parallel Insertion method in controlling the violation of due-dates. Similar to the sequential case, the Systematic Slack Costing rule is able to reduce the travelling cost, but it is not able to control the violation of due-dates of the products.
- Results shows that, the behaviour of the algorithms and the task despatch rules, are not changing by varying the number of vehicles available at the depot .
- Results shows that, by increasing the weightage for the penalty cost with respect to the travelling cost is able to control the violation of due-dates of the products till some value, then the algorithm gets saturated and gave the same results by further increase of the weightage.

Comparision of the performance of all heuristics interms of %dev. of total travelling cost with respect to the best one

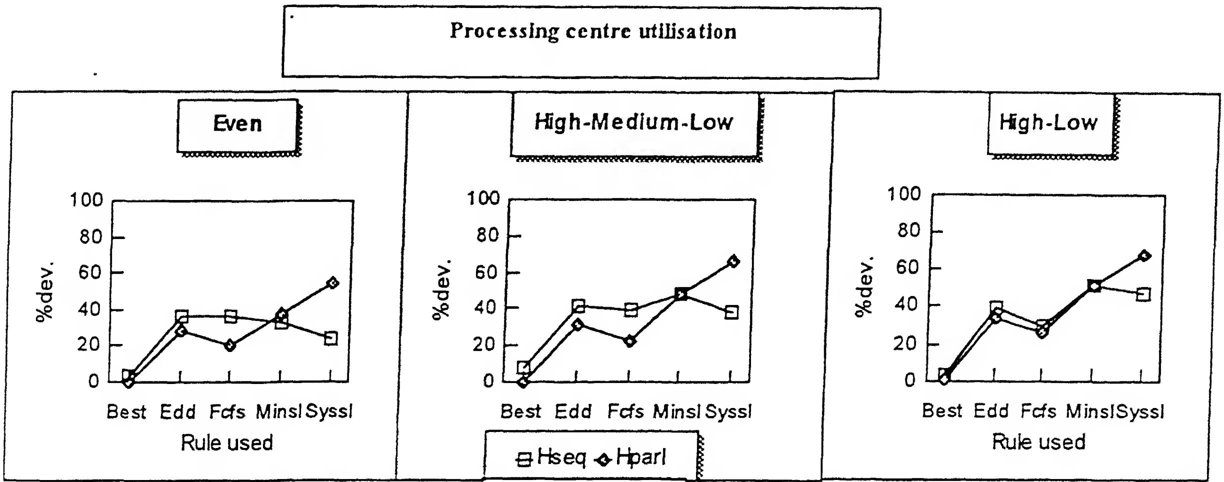


Fig 4.2.1

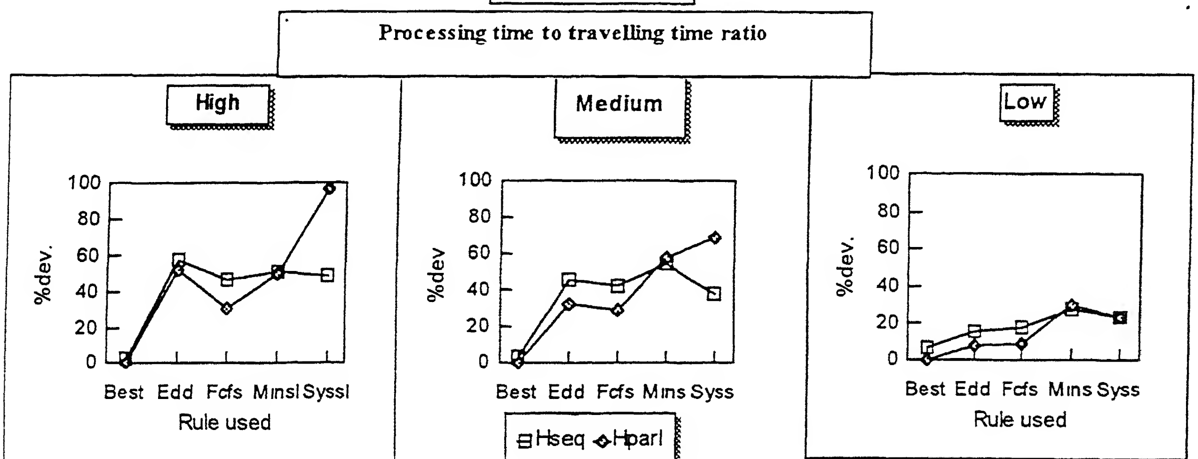


Fig 4.2.2

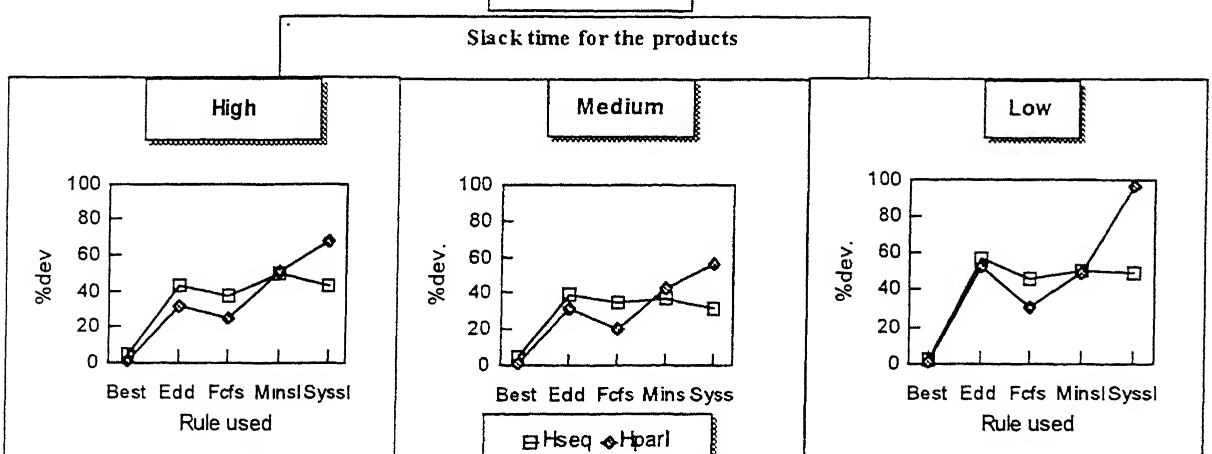


Fig 4.2.3

Fig 4.2

Comparision of the performance of all heuristics interms of number of vehicles used

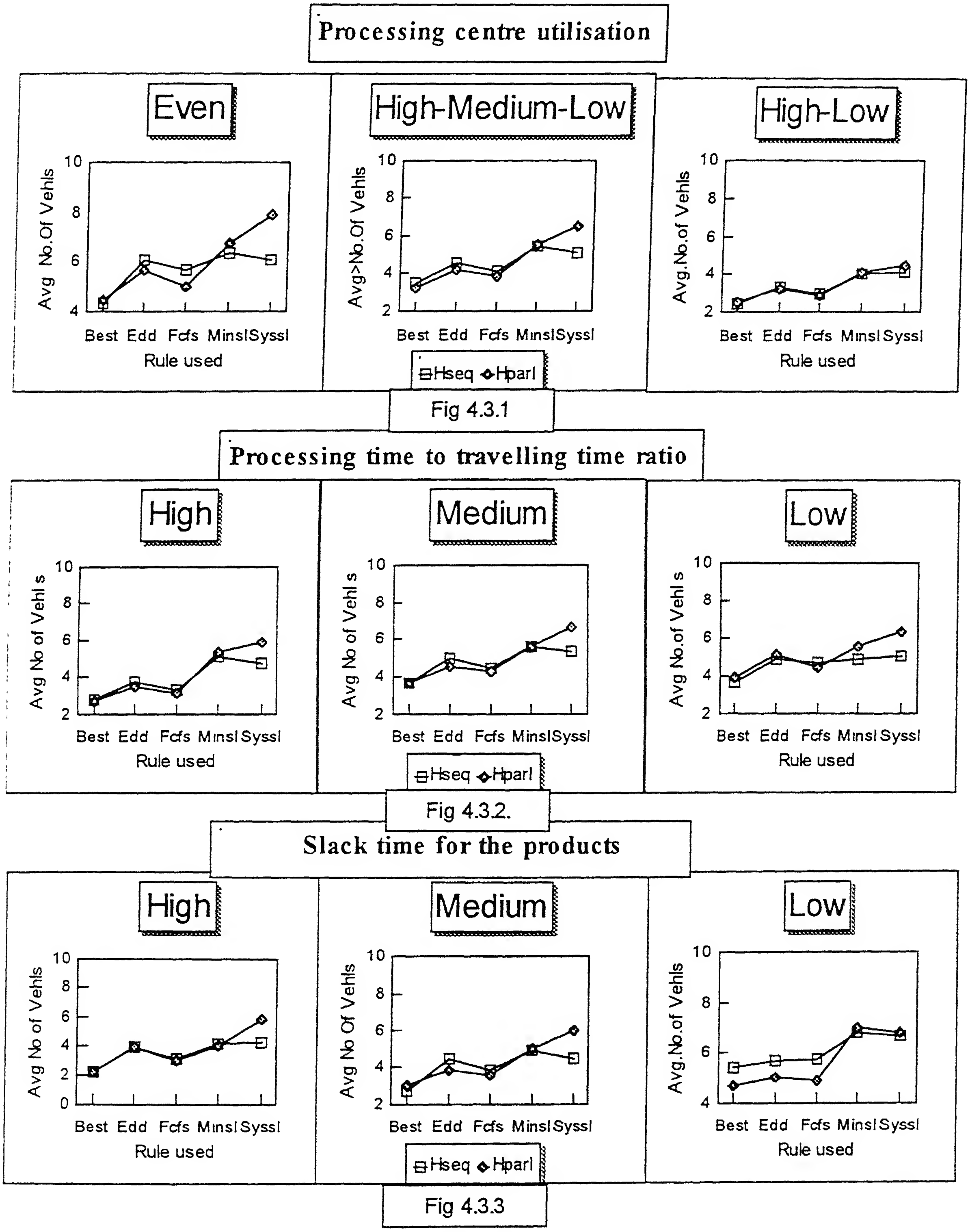


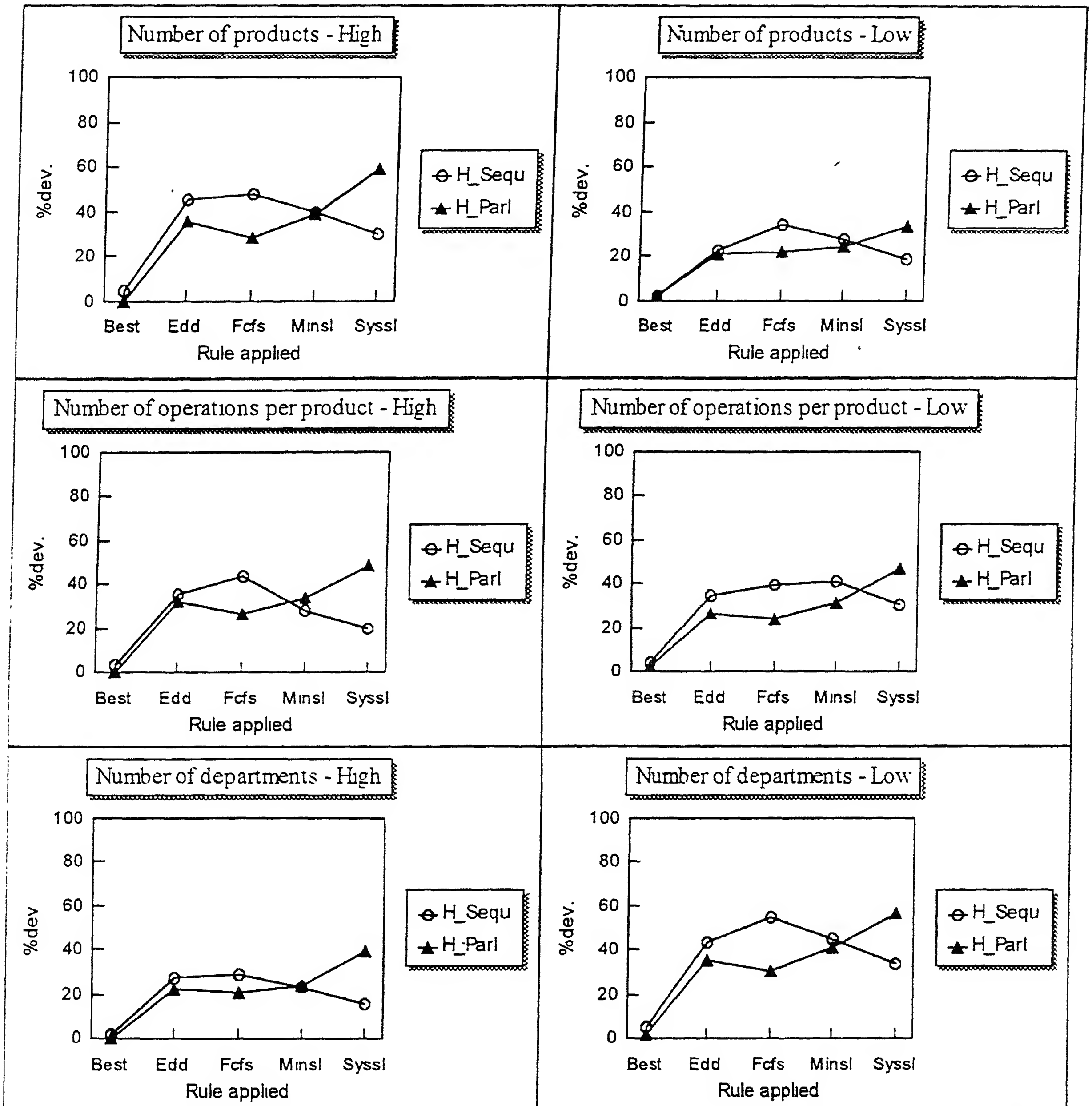
Fig 4.3

Computational performance of all the heuristics interms of travelling cost with respect to the best one on the problems generated by varying the process data of the problems

	Seq_Best		Parl_Best		Seq Edd	Parl Edd	Seq Fcfs	Parl Fcfs	Seq Minsl	Parl Minsl	Seq Syssl	Parl Syssl
	avg. %Dev from best	No.of best prob.'s	avg. %Dev from best	No.of best prob.'s	avg %Dev from best	avg. %Dev from best	avg. %Dev from best	avg. %Dev from best	avg. %Dev from best	avg. %Dev from best	avg. %Dev from best	avg. %Dev from best
Pocessing centre utilisation												
Even	2.96	3	0.6	6	37	28.5	36.3	20.1	32.7	37.7	24.5	54.71
High Medium Low	7.52	1	0.03	8	42	31.1	39.82	22	48.2	48.8	37.74	66
High Low	3.35	4	1.43	5	39.7	33.5	29.45	26.3	51.8	51.1	47.4	68.2
Processing time to travelling time ratio												
High	4	3	0.53	6	42.8	31 8	37.2	25	50.2	51.3	42.82	67.8
Medium	5	3	0.47	6	36.35	29.6	33.7	22.9	45.2	44.06	35.8	65
Low	4.6	2	1.02	7	39.5	31.7	34.6	20.4	37.25	42.16	31	56.2
Slack time available												
High	2.4	4	1.1	5	56.9	52.24	45.75	30.33	50.8	49.7	49.05	96.1
Medium	4.2	4	0.47	5	46 1	32.4	42 25	29.4	54.5	57.7	37.6	69.3
Low	7.25	1	0.45	8	15.7	8.5	17.6	8.5	27 3	30.11	22.98	23.6

Table 4.3

Comparision of the perfomance of all heuristics in terms of %dev. of total travelling cost with respect to the best one



Y-axis lable :

Avg. % deviation of the total cost of a heuristic from the minium total cost among all the heuristics

Comparision of the performance of all the heuristics in terms of avg. number of vehicles used

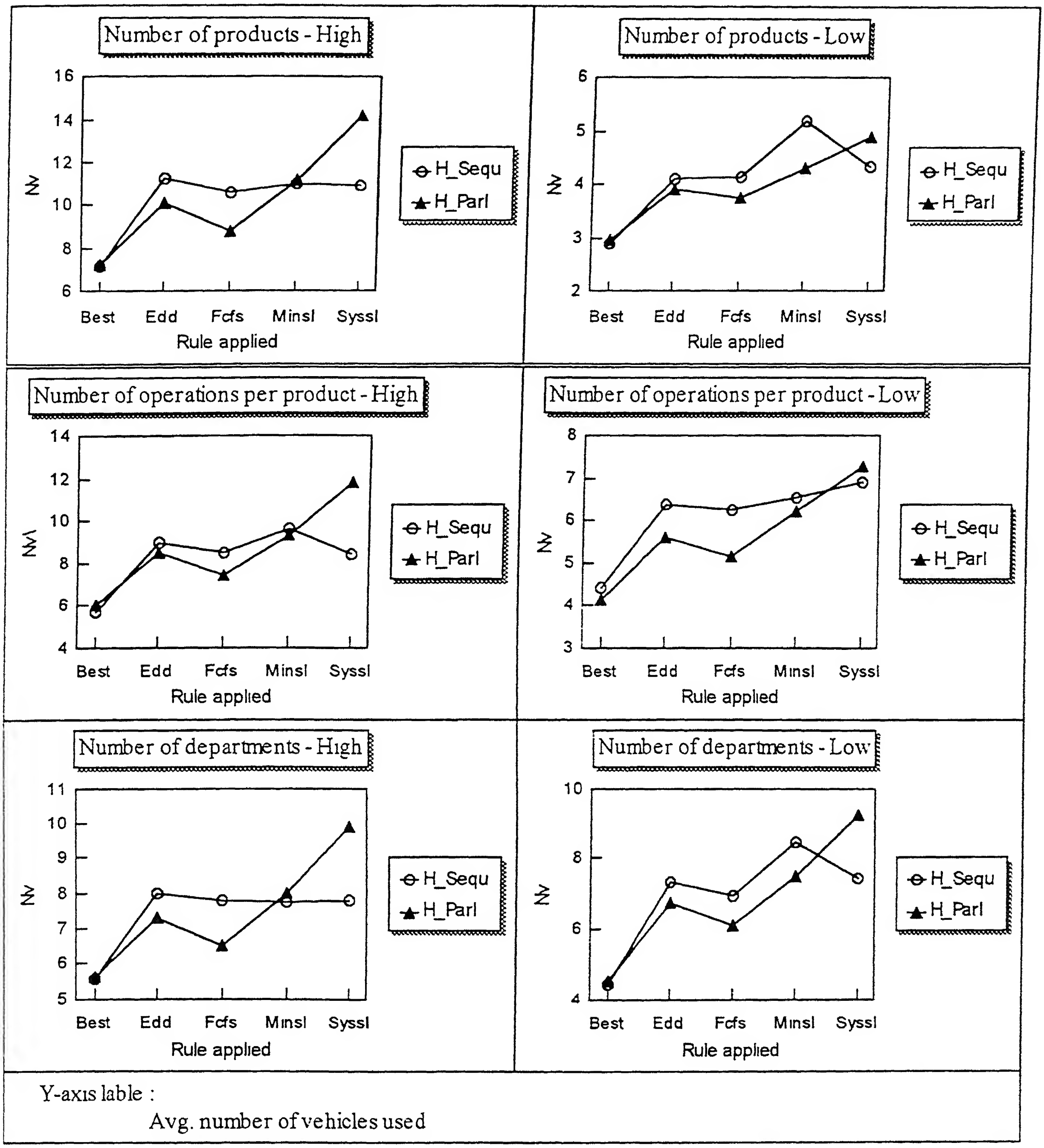
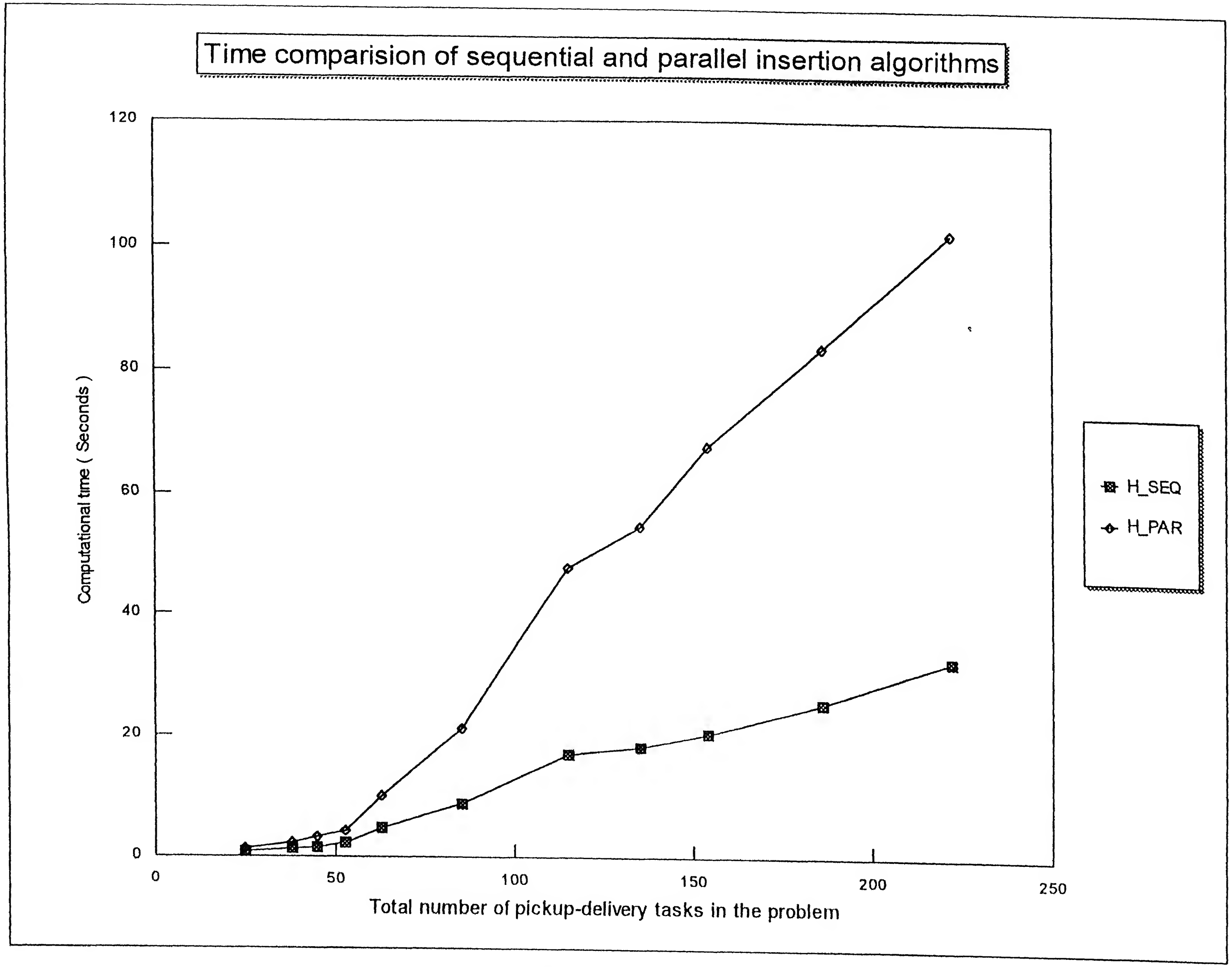


Fig 4.5

Comparision of the performance of all the heuristics interms of total travelling cost with respect to best one on the problems generated by varying size perameters

	Seq_Best		Parl_Best		Seq Edd	Parl Edd	Seq Fcfs	Parl Fcfs	Seq Minsl	Parl Minsl	Seq Syssl	Parl Syssl
	%dev. From best	No. Of best prob.'s out of 20	%dev. From best	No. Of best prob.'s out of 20	%dev. From best	No. Of best prob.'s out of 20	%dev. From best	No. Of best prob.'s out of 20	%dev. From best	No. Of best prob.'s out of 20	%dev. From best	No. Of best prob.'s out of 20
Number of products												
High	4.5	3	0.07	17	45.34	35.72	47.8	28	39.4	39.4	29.7	59.4
Low	2.5	3	0.56	17	22.7	20.66	33.8	21.85	27.2	24	18.5	33.67
Number of Open.s per product												
High	3.16	4	0.09	16	35.2	31.6	43.15	26.5	27.44	33.2	19.3	48.3
Low	4	2	0.51	18	34.7	25.82	39.54	23.8	40.52	31.41	30.07	46.85
Number of Process centres												
High	1.82	4	0 .11	16	27.2	22.4	28.7	20.3	23.4	39	15.9	39
Low	5.4	1	0.5	19	43.1	35.5	54.8	30.32	44 8	40.9	33.7	56.6

Table 4.4



Time comparision of sequential and parallel insertion algorithms
Computational time (Seconds)
120
100
80
60
40
20
0
0
50
100
150
200
250
Total number of pickup-delivery tasks in the problem
H_SEQ
H_PAR

Fig 4.6

Sequential insertion procedure for restricted number of vehicles case

Number of vehicles available is varied												
		Best values of slack and vehicle waiting cost		EDD		FCFS		MINSL		SYSSL		
Num. Of Veh.	Relative wt. Of penalty	Total travel Cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total duedate shift	Total travel cost	Total duedate shift
1	1.00	7337	417.8	8591	269.04	9035	465.1	7943	264.10	8111.0	617.35
2	1.00	7485	92.77	9345	98.05	9319	141.1	9187	59.58	7749.0	213.86
3	1.00	7140	0.00	9669	45.67	9451	33.45	8445	12.78	7294.0	72.94
4	1.00	6488	0.00	9030	9.22	9043	9.35	8075	12.78	7233.0	36.60

Table 4.5

Relative weight of penalty cost with respect to travelling cost is varied											
		Best values of slack and vehicle waiting cost		EDD		FCFS		MINSL		SYSSL	
No.of vehicles	Rel. Wt Of pen	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift
2	0.10	6526.00	160.20	874200	89.56	9004	143 22	8092.0	39.06	7486.0	208.14
2	0.20	7322.00	109.08	9149.00	94.23	9107.00	142.37	8583.00	54.19	7640.0	214.47
2	0.30	7485.00	97.45	9149.00	94.23	9281.00	143.03	8583.00	54.19	7749.0	213.86
2	0.40	7485.00	92.77	9149.00	94.23	9319.00	141.06	8583.00	54.19	7749.0	213.86
2	0.50	7485.00	92.77	9320.00	95.18	9319.00	141.06	8708.00	54.99	7749.0	213.86
2	0.60	7485.00	92.77	9320.00	95.18	9319 00	141.06	9187.00	59.58	7749.0	213.86
2	1.00	7485.00	92.77	9335.00	98.05	9319.00	141.06	9187.00	59.58	7749.0	213.86
2	2.00	7485.00	92.77	9586.00	101.12	9319.00	141.06	9121.00	60.51	7749.0	213.86

Table 4.6

Parallel insertion procedure for restricted number of vehicles case

Number of vehicles available is varied											
		Best value of Slack and vehicle waiting cost		EDD		FCFS		MINSL		SYSSL	
Num. Of Veh.	Relative wt. Of penalty	Total travel Cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total duedate shift	Total travel cost	Total duedate shift
1	1.00	7334	417.8	11177	541.04	13454	822.1	8213	708.90	7420.0	550.03
2	1.00	7410	74.62	9758	109.79	11738	549.1	8041	309.01	7165.0	303.24
3	1.00	6650	0.00	9104	37.12	11679	452.2	8094	114.58	6773.0	162.50
4	1.00	6479	0.00	8127	17.21	10608	261.5	8000	102.92	6901.0	99.60

Table 4.7

Relative weight of penalty cost with respect to travelling cost is varied											
No of vehicles	Rel. Wt. Of pen	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift	Total travel cost	Total shift
2	0.10	6282.00	148.18	8733.00	80.33	11738.0	549.14	7965.00	319.19	7047.0	307.27
2	0.20	6675.00	84.20	9421.00	118.54	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	0.30	6778.00	76.20	9854.00	109.29	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	0.40	7410.00	74.62	9645.00	115.29	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	0.50	7410.00	74.62	9758.00	109.79	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	0.60	7410.00	74.62	9758.00	109.79	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	1.00	7410.00	74.62	9758.00	109.79	11738.0	549.14	8041.00	309.01	7165.0	303.24
2	2.00	7410.00	74.62	9726.00	114.82	11738.0	549.14	8041.00	309.01	7165 0	303.24

Table 4.8

CHAPTER 5

CONCLUSIONS AND AVENUES FOR FUTURE WORK

5.1 Conclusions :

In this dissertation a vehicle routing and scheduling problem in a multi-product distributed manufacturing system is modelled as a standard pickup-delivery problem with time windows and additional product sequence constraint. We formulated this problem as a mixed integer program and discussed the complexity and inability of the exact methods to solve the practical size problems We proposed two approximate algorithms based on sequential and parallel insertion procedures and tested few despatch rules for assigning the tasks to the vehicles. We performed computational study by designing various types of problems.

The computational study shows that at the best values of slack cost and vehicle waiting cost the Parallel Insertion Heuristic is performing always better than Sequential Insertion Heuristic. Among the despatch rules applied, the First Come First Serve rule applied to Parallel Insertion procedure is performing better in almost all of the cases. In some of the cases Systematic Slack Costing rule applied to Sequential Insertion procedure is performing well. In terms of average number of vehicles used the First Come First Serve despatch rule is performing well for both the algorithms.

5.2 Avenues of future work:

In this work, we assumed there is no capacity restriction for the process centres. One can extend this work, by considering capacity restrictions in the process centres. Advanced search techniques like genetic algorithms and artificial intelligence techniques can also be tried on this problem.

APPENDIX-A

PROCESS DATA FACTORS COMBINATIONS WHILE GENERATING PROBLEMS

	Process utilisation	Pr.time/Tr.time	Slack available
1	Even	High	High
2	Even	High	Medium
3	Even	High	Low
4	Even	Medium	High
5	Even	Medium	Medium
6	Even	Medium	Low
7	Even	Low	High
8	Even	Low	Medium
9	Even	Low	Low
10	High-Medium-Low	High	High
11	High-Medium-Low	High	Medium
12	High-Medium-Low	High	Low
13	High-Medium-Low	Medium	High
14	High-Medium-Low	Medium	Medium
15	High-Medium-Low	Medium	Low
16	High-Medium-Low	Low	High
17	High-Medium-Low	Low	Medium
18	High-Medium-Low	Low	Low
19	High-Low	High	High
20	High-Low	High	Medium
21	High-Low	High	Low
22	High-Low	Medium	High
23	High-Low	Medium	Medium
24	High-Low	Medium	Low
25	High-Low	Low	High
26	High-Low	Low	Medium
27	High-Low	Low	Low

APPENDIX - B

PROBLEM SIZE FACTORS COMBINATION WHILE GENERATING PROBLEMS

	No. of products	No. of operations per product	No. of process centres
1	High	High	High
2	High	High	Low
3	High	Low	High
4	HIgh	Low	Low
5	Low	High	High
6	Low	High	Low
7	Low	Low	High
8	Low	Low	Low

REFERENCES

(1) ASSAD, A.A. and GOLDEN, B.L., eds. [1988], "Vehicle routing : Methods and Studies", North Holland , New York, N.Y.

(2) BALAKRISHNAN, N., [1993], "Simple Heuristics for the Vehicle Routeing Problem with Soft Time Windows", *Journal of the Operations Research Society*, Vol.44, pp 279-287.

(3) BALL, M.O., MAGNANTI, T.L., MONMA, C.L. and NEMHAUSER .G.L., eds.[1995], "Network Routing", *Handbooks in Operations Research and Management Science,* Vol.8. North Holland.

(4) BODIN, L.D., GOLDEN. B.L., ASSAD, A.A. and BALL, M.O.[1983], " Routing and scheduling of Vehicles and Crews . The State of Art", *Computers and Operations Research,* Vol.10, pp. 69-211.

(5) CHRISTOFIDES. N., [1985], "Vehicle routing", *The Travelling Salesmen Problem, A guided tour of Combinatorial optimisation.* E.L.LAWLER etal.(Eds.), Wiley, Chichester, pp. 431-448.

(6) CHRISTOFIDES. N., MINGOZZI. A.and TOTH. P.,[1981a], " Exact algorithms for the vehicle routing problem. based on spanning tree and shortest path relaxations. *Math. Programming 20,* pp 255-282.

(7) CLARKE, G. and WRIGHT .J.W. [1964], " Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research,* Vol.12, pp. 568-581.

(8) DESROCHERS, M. and SOUMIS, F., [1989], " A Column Generation Approach to the Urban Transit Crew Scheduling Problem", *Transportation Science,* Vol. 23, pp. 1-13.

(9) DESROSIERS, J., DUMAS, Y. and SOUMIS, F.,[1986b], " A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time-windows", *Amer. J. Math. Management Science,* Vol.6, pp. 301-326.

(10) DESROSIERS, J., SOUMIS, M., and DESROCHERS, M., [1984], " Routing with Time Windows by Column Generation", *Networks,* Vol.14, pp. 545-565.

(11) DUMAS, Y., DESROSIERS, J. and SOUMIS, F., [1991], "The pickup and delivery problem with time windows", *European Journal of Operations Research,* Vol.54, pp. 7-22.

(12) FISHER, M.L., JAIKUMAR, R., [1981], "A Generalised Assignment Heuristic for Vehicle Routing", *Networks,* Vol.11, pp. 109-124.

(13) GENDREAU, M., HERTZ, A. and LAPORTE, G., [1994], "A Tabu Search Heuristic for the Vehicle Routing Problem", *Management Science,* Vol.40, pp. 1276-1290.

(14) GILLET, B. and MILLER, L. [1974], " A Heuristic Algorithm For the Vehicle Dispatching Problem," *Operations Research,* Vol.22, pp. 340-349.

(15) HARKER, P.T., [1990], "The use of advanced train controlsystem in scheduling and operating rail roads: models, algorithms and applications, in: *Transportation Research Record,* pp. 101-110.

(16) JAW, J.J., ODONI, R., PSARAFTIS, H.N. and WILSON, H.M., [1986], "A Heuristic algorithm for Multi-Vehicle Advance Request Dial-A-Ride Problem with Time Windows", *Transportation Research-B,* Vol.20B, pp. 243-257.

(17) KOSKOSIDIS, Y.A., POWELL, W.B. and SOLOMON, M.M., [1992], " An Optimisation -Based Heuristic for Vehicle Routing and Scheduling with Soft Time Window Constraints", *Transportation Science,* Vol. 26, pp. 69-85.

(18) LAPORTE, G.[1992], " The Vehicle Routing Problem : An overview of Exact and Approximate Algorithms,"*European J. Operations Res.,* Vol.59, pp. 345-358.

(19) LOVOI, S., MINOUX, M. and ODIER, E., [1988], " A new approach for crew pairiing problems by column generation with an application to air transportation ", *European J.of Opern. Research,* Vol.35, pp. 45-58.

(20) MINGOZZI, A.and TOTH, P.,[1979] " The Vehicle Routing Problem", *Combinatorial Optimisation,* N.CHRISTOFIDES etal. (Eds.), Wiley, Chichester, pp. 315-338.

(21) PSARAFTIS, H. [1983], " An exact for a single vehicle Many-to-Many Dial-A-Ride Problem with Time Windows," *Transportation Science*, Vol.17, pp. 315-358.

(22) SEXTON, T. and BODIN, L. [1985a], " Optimisation single vehicle Many-to-Many Operations with Desired Delivery Times: I. Scheduling." *Transportation Science,* Vol.19, pp. 378-410.

(23) SOLOMON, M.M., [1987], " Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints", *Operations Research*, Vol.35, pp. 254-265.